Multi-input Functional Encryption*

Shafi Goldwasser[1,**], S. Dov Gordon[2], Vipul Goyal[3], Abhishek Jain[4], Jonathan Katz[5,***], Feng-Hao Liu[5], Amit Sahai[7,†], Elaine Shi[5,‡], and Hong-Sheng Zhou[9,§]

[1] MIT and Weizmann, Israel
shafi@csail.mit.edu
[2] Applied Communication Sciences, USA
sgordon@appcomsci.com
[3] Microsoft Research, India
vipul@microsoft.com
[4] Boston University and MIT, USA
abhishek@csail.mit.edu
[5] University of Maryland, USA
{jkatz,fenghao,elaine}@cs.umd.edu
[6] UCLA
sahai@cs.ucla.edu
[7] Virginia Commonwealth University, USA
hszhou@vcu.edu

* This conference proceedings publication is the result of a merge of two independent and concurrent works. The two papers were authored by Goldwasser, Goyal, Jain, and Sahai; and by Gordon, Katz, Liu, Shi, and Zhou.

** Research supported by NSFEAGER award # CNS1347364 DARPA award # FA8750-11-2-0225 and the Simons Foundation - Investigation Award.

*** Research supported by NSF awards #1111599 and #1223623, and by the US Army Research Laboratory and the UK Ministry of Defence under Agreement Number W911NF-06-3-0001. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the U.S. Government, the UK Ministry of Defense, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

† Research supported in part from a DARPA/ONR PROCEED award, NSF grants 1228984, 1136174, 1118096, and 1065276, a Xerox Faculty Research Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014-11- 1-0389. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government.

‡ This work is partially supported by NSF award CNS-1314857 and a Google Research Award.

§ This work is partially supported by an NSF CI postdoctoral fellowship, and was mostly done while at the University of Maryland.

P.Q. Nguyen and E. Oswald (Eds.): EUROCRYPT 2014, LNCS 8441, pp. 578–602, 2014.

Abstract. We introduce the problem of Multi-Input Functional Encryption, where a secret key sk_f can correspond to an n-ary function f that takes multiple ciphertexts as input. We formulate both indistinguishability-based and simulation-based definitions of security for this notion, and show close connections with indistinguishability and virtual black-box definitions of obfuscation.

Assuming indistinguishability obfuscation for circuits, we present constructions achieving indistinguishability security for a large class of settings. We show how to modify this construction to achieve simulation-based security as well, in those settings where simulation security is possible.

1 Introduction

Traditionally, encryption has been used to secure a communication channel between a unique sender-receiver pair. In recent years, however, our networked world has opened up a large number of new usage scenarios for encryption. For example, a single piece of encrypted data, perhaps stored in an untrusted cloud, may need to be used in different ways by different users. To address this issue, the notion of functional encryption (FE) was developed in a sequence of works [19,13,7,14,15,6,16,18]. In functional encryption, a secret key sk_f can be created for any functions f from a class $\mathcal{F}$; such a secret key is derived from the master secret key MSK. Given any ciphertext c with underlying plaintext x, using SK_f a user can efficiently compute $f(x)$. The security of FE requires that the adversary "does not learn anything" about x, other than the computation result $f(x)$.

How to define "does not learn anything about" x is a fascinating question which has been addressed by a number of papers, with general formal definitions first appearing in [6,16]. The definitions range from requiring a strict *simulation* of the view of the adversary, which enlarges the range of applications, but has been shown to necessitate a ciphertext whose size grows with the number of functions for which secret keys will ever be released [1] (or a secret key whose size grows with the number of ciphertexts that will ever be released [6]), to an *indistinguishability* of ciphertexts requirement which supports the release of an unbounded number of function keys and short ciphertexts.

Functional encryption seems to offer the perfect non-interactive solution to many problems which arise in the context of delegating services to outside servers. A typical example is the delegation of spam filtering to an outside server as follows: Alice publishes her public key online and gives the spam filter a key for the filtering function; Users sending email to Alice will encrypt the email with her public key. The spam filter can now determine by itself, for each email, whether to pass it along to Alice's mailbox or to deem it as spam, but without ever learning anything about Alice's email (other than the fact that it was deemed a spam message or not). This example inherently requires computing a function f on a single ciphertext.

Multi-Input Functional Encryption. It is less clear, however, how to define or achieve functional encryption in the context of computing a function defined

over *multiple* plaintexts given their corresponding ciphertexts, or further, the computation of functions defined over plaintexts given their ciphertexts each encrypted under a different key. Yet, these settings, which we formalize as *Multi-Input Functional Encryption*, encompass a vast landscape of applications, going way beyond delegating computation to an untrusted server or cloud.

Let us begin by clarifying the setting of Multi-Input Functional Encryption: Let f be an n-ary function where $n > 1$ can be a polynomial in the security parameter. We begin by defining multi-input functional encryption where the owner of a master secret key MSK can derive special keys SK_f whose knowledge enables the computation of $f(x_1, \ldots, x_n)$ from n ciphertexts $c_1, \ldots, c_n$ of underlying messages $x_1, \ldots, x_n$ with respect to the same master secret key MSK. We next allow the different ciphertexts c_i to be each encrypted under a different encryption key EK_i to capture the setting in which each ciphertext was generated by an entirely different party.

Let us illustrate a few settings in which one would want to compute a function over multiple plaintexts given the corresponding ciphertexts.

Example: Multi-input symmetric-key FE can be used for secure searching over encrypted data, where it can function in the same role as *order-preserving encryption* (OPE) [4,5] or, more generally, property-preserving encryption [17]. A direct application of our construction yields the first OPE scheme to satisfy the indistinguishability notion of security proposed by Boldyreva et al. [4], resolving a primary open question in that line of research. More specifically, consider a setting in which a client uploads several encrypted data items $c_1 = \mathsf{Enc}(x_1)$, $\ldots$, $c_n = \mathsf{Enc}(x_n)$ 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 $c^* = \mathsf{Enc}(t)$ along with a secret key SK_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. In fact, we can hide even more information than OPE: if the client tags every data item with a '0' (i.e., uploads $c_i = \mathsf{Enc}(0\|x_i)$) and tags the search term with a '1' (i.e., sends $c^* = \mathsf{Enc}(1\|t)$), then the client can send SK_f for the function

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

Thus, SK_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 followed to enable arbitrary searches over encrypted data while revealing only a minimal amount of information. We note also that the search query itself can remain hidden as well.

More generally, suppose Alice wishes to perform a certain class of general SQL queries over this database. If we use ordinary functional encryption, Alice would need to obtain a separate secret key for every possible valid SQL query, a potentially exponentially large set. Multi-input functional encryption allows

us to address this problem in a flexible way. We highlight two aspects of how Multi-Input Functional Encryption can apply to this example:

- Let f be the function where $f(q, x)$ first checks if q is a valid SQL query from the allowed class, and if so $f(q, x)$ is the output of the query q on the database x. Now, if we give the secret key SK_f *and* the encryption key ek_1 to Alice, then Alice can choose a valid query q and encrypt it under her encryption key EK_1 to obtain ciphertext c_1. Then she could use her secret key SK_f on ciphertexts c_1 and c_2, where c_2 is the encrypted database, to obtain the results of the SQL query.
- Furthermore, if our application demanded that multiple users add or manipulate different entries in the database, the most natural way to build such a database would be to have different ciphertexts for each entry in the database. In this case, for a database of size n, we could let f be an $(n+1)$-ary function where $f(q, x_1, \ldots, x_n)$ is the result of a (valid) SQL query q on the database $(x_1, \ldots, x_n)$.

1.1 This Paper

This paper is a merge of two independent works, both of which can be found online [10,12]. These two works contain many overlapping results dedicated to the study of multi-input functional encryption, starting with formalizations of security. In them, the authors provide both feasibility results and negative results with respect to different definitions of security. Following the single-input setting, they consider two notions of security, namely, indistinguishability-based security (or IND security for short) and simulation-based security (or SIM security for short). Below we summarize only what appears in this proceedings, and refer the reader to the full versions for a more complete study of the subject.

Indistinguishability-Based Security. We start by considering the notion of indistinguishability-based security for n-ary multi-input functional encryption: Informally speaking, in indistinguishability security for multi-input functional encryption, we consider a game between a judge and an adversary. First, the judge generates the master secret key MSK, evaluation keys $\{\mathsf{EK}_1, \ldots, \mathsf{EK}_n\}$, and public parameters, and gives to the adversary the public parameters and a subset of evaluation keys (chosen by the adversary). Then the adversary can request any number of secret keys SK_f for functions f of the adversary's choice. Next, the adversary declares two "challenge vectors" of sets X^0 and X^1, where X_i^b is a set of plaintexts. The judge chooses a bit b at random, and for each $i \in [n]$, the judge encrypts every element of X_i^b using evaluation key ek_i to obtain a tuple of "challenge ciphertexts" C, which is given to the adversary. After this, the adversary can again request any number of secret keys SK_f for functions f of the adversary's choice. Finally, the adversary has to guess the bit b that the judge chose.

If the adversary has requested any secret key SK_f such that there exist vectors of plaintexts x^0 and x^1 where for every $i \in [n]$, either $x_i^b \in X_i^b$ or the adversary has EK_i, such that $f(x^0) \neq f(x^1)$, then we say that this function f

splits the challenge, and in this case the adversary loses – because the legitimate functionalities that he has access to already allow him to distinguish between the scenario where $b = 0$ and $b = 1$. If the adversary has never asked for any splitting function, and nevertheless the adversary guesses b correctly, we say that he wins. The indistinguishability-based security definition requires that the adversary's probability of winning be at most negligibly greater than $\frac{1}{2}$.

This definition generalizes the indistinguishability-based definition of (single-input) functional encryption, which was historically the first security formalization considered for functional encryption [19]. Informally speaking, this definition captures an information-theoretic flavor of security, where the adversary should not learn anything beyond what is information-theoretically revealed by the function outputs it can obtain.

With regard to the indistinguishability notion of security, we obtain the following results:

- **Indistinguishability-based security implies indistinguishability obfuscation, even for single-key security.** We show that the existence of a multi-input functional encryption scheme achieving indistinguishability-based security for all circuits implies the existence of an indistinguishability obfuscator [2] for all circuits, *even when security is only needed against an adversary that obtain a single secret key*, and where the adversary does not receive any evaluation keys. This stands in stark contrast to the single-input setting, where [18] showed how to obtain single-key secure (single input) functional encryption for all circuits, under only the assumption that public-key encryption exists. Indeed further research in single-key security for functional encryption has largely focused on efficiency issues [11] such as succinctness of ciphertexts, that enable new applications. In the setting of multi-input security, in contrast, even single key security must rely on the existence of indistinguishability obfuscation.
- **Positive Result, General Setting.** On the other hand, if we assume that an indistinguishability obfuscator for general circuits exists with sub-exponential security (the first candidate construction was recently put forward by [9]), and we assume that sub-exponentially secure one-way functions exist, then we obtain full indistinguishability-based security for any polynomial-size challenge vectors, with any subset of evaluation keys given to the adversary. Furthermore, our construction has security when the adversary can obtain any *unbounded* polynomial number of secret keys SK_f. Our result is obtained by first achieving *selective* security, where the adversary must begin by declaring the challenge vectors, using indistinguishability obfuscation and one-way functions (leveraging the results of [20]). Then we use complexity leveraging to obtain full security in a standard manner.
- **Positive Result, Symmetric Key Setting.** We consider a special case where the adversary is not given any of the evaluation keys, corresponding to a typical symmetric key setting. As an example, this can be useful in a scenario where a single user wishes to outsource their private dataset to

one or more untrusted servers, issuing keys to facilitate searches over the data. In this setting, we give a construction with succinct ciphertexts whose size is dependent only on the security parameter, and not on the number of challenge plaintexts. We remark that the size of the public parameters, which are used in encryption and decryption, still grows with the number of challenge plaintexts. We refer the reader to the full versions [10,12] for ways to remove this dependency.

– **Positive Result, Isolated Time-steps.** We consider one last setting where each party encrypts a single plaintext in every *time-step*, and we modify the security definition to prevent the computation on ciphertexts from different time-steps. In this setting, as above, the adversary can request any subset of the evaluation keys. We give a construction that has succinct ciphertexts, dependent only on the security parameter, and independent of the number of time-steps or the number of challenge plaintexts. (The remark about the public parameters that appears above still applies.)

Simulation-Based Security. In simulation-based security, informally speaking, we require that every adversary can be simulated using only oracle access to the functions f for which the adversary obtains secret keys, even when it can obtain a set of "challenge" ciphertexts corresponding to unknown plaintexts – about which the simulator can only learn information by querying the function f at these unknown plaintexts. We highlight two natural settings for the study of simulation security for multi-input functional encryption: (1) the setting where an adversary has access to an encryption key (analogous to the public-key setting), and (2) the setting where the adversary does not have access to any encryption keys (analogous to the secret key setting). The security guarantees which are achievable in these settings will be vastly different as illustrated below.

Several works [6,1,3] have shown limitations on parameters with respect to which simulation-based security can be achieved for single-input functional encryption. For multi-input functional encryption, due to the connection to obfuscation discussed above, the situation for simulation-based security is more problematic. Indeed, it has been a folklore belief that n-ary functional encryption with simulation-based security would imply Virtual Black-Box obfuscation, which is known to be impossible [2]. We strengthen and formalize this folklore in three results:

– In the setting where the adversary receives only a single key for a single n-ary function, and receives no evaluation keys, and where the adversary can obtain a set of challenge ciphertexts that can (informally speaking) form a super-polynomial number of potential inputs to f, if simulation security is possible then virtual black-box obfuscation must be possible for arbitrary circuits, which is known to be impossible [2]. This follows immediately from the same construction that shows the connection of indistinguishability-based security to indistinguishability obfuscation mentioned above, and most directly formalizes the folklore belief mentioned above.
– In the setting where the adversary receives only a single key for a 2-ary function, and receives one evaluation key and one challenge ciphertext, if

simulation security is possible then virtual black-box obfuscation must be possible for arbitrary circuits, which is known to be impossible [2].
- The above results demonstrate that we cannot achieve simulation security for arbitrary multi-input functions. Looking at which functions we might support, we define a new notion of *learnable functions* and demonstrate that we can only achieve simulation-based security for this type of function. Informally, we call a 2-ary function, $f(\cdot,\cdot)$, *learnable* if, when given a description of f and oracle access to $f(x,\cdot)$, one can output the description of a function that is indistinguishable from $f_x(\cdot)$ (i.e. from the function obtained when restricting f to input x).

Positive Result. In light of these negative results, the only hope for obtaining a positive result lies in a situation where: (1) no evaluation keys are given to the adversary, and (2) the challenge ciphertexts given to the adversary can only form a polynomial number of potential inputs to valid functions. Assuming one-way functions and indistinguishability obfuscation, for any fixed polynomial bound on the size of these potential inputs we give a construction that achieves simulation-based security for multi-input functional encryption where the adversary obtains no evaluation keys, but can obtain some fixed polynomial number of secret keys SK_f before obtaining challenge ciphertexts, as well as an unbounded number of secret keys SK_f after obtaining challenge ciphertexts.

Finally, we complement this positive result by showing that even in the setting where the adversary obtains no evaluation keys and an unlimited number of challenge ciphertexts, simulation-based security is impossible if the adversary can ask for even one secret key SK_f before it obtains the challenge ciphertexts.

Our Techniques. We have several results in this work, but to provide some flavor of the kinds of difficulties that arise in the multi-input functional encryption setting, we now describe some of the issues that we deal with in the context of our positive result for indistinguishability-based security for multi-input functional encryption. (Similar issues arise in our other positive result for simulation-based security.)

The starting point for our construction and analysis is the recent single-input functional encryption scheme for general circuits based on indistinguishability obfuscation due to [9]. However, the central issue that we must deal with is one that does not arise in their context: Recall that in the indistinguishability security game, the adversary is allowed to get secret keys for any function f, as long as this function does not "split" the challenge vectors $\boldsymbol{X}^0$ and $\boldsymbol{X}^1$. That is, as long as it is *not* the case that there exist vectors of plaintexts $\boldsymbol{x}^0$ and $\boldsymbol{x}^1$ where for every $i \in [n]$, either there exists j such that $x_i^b \in X_j^b$ or the adversary has EK_i, such that $f(x^0) \neq f(x^1)$. A crucial point here is what happens for an index i where the adversary does *not* have EK_i. Let us consider an example with a 3-ary function, where the adversary has EK_1, but neither EK_2 nor EK_3.

Suppose the challenge ciphertexts $(\mathsf{CT}_1, \mathsf{CT}_2, \mathsf{CT}_3)$ are encryptions of either (y_1^0, y_2^0, y_3^0) or (y_1^1, y_2^1, y_3^1). Now, any function f that the adversary queries is required to be such that $f(\cdot, y_2^0, y_3^0) \equiv f(\cdot, y_2^1, y_3^1)$ and $f(y_1^0, y_2^0, y_3^0) = f(y_1^1, y_2^1, y_3^1)$.

However, there may exist an input plaintext (say) z such that $f(y_1^0, y_2^0, z) \neq f(y_1^1, y_2^1, z)$. This is not "supposed" to be a problem because the adversary does not have EK_3, and therefore it cannot actually query f with z as its third argument.

However, in the obfuscation-based approach to functional encryption of [9] that we build on, the secret key for f is essentially built on top of an obfuscation of f. Let CT^* denote an encryption of z w.r.t. EK_3. Then, informally speaking, in one of our hybrid experiments, we will need to move from an obfuscation that on input $(\mathsf{CT}_1, \mathsf{CT}_2, \mathsf{CT}^*)$ would yield the output $f(y_1^0, y_2^0, z)$ to another obfuscation that on the same input would yield the output $f(y_1^1, y_2^1, z)$. Again, while an adversary may not be able explicitly perform such a decryption query, since we are building upon *indistinguishability obfuscation* – which only guarantees that obfuscations of circuits that implement *identical* functions are indistinguishable – such a hybrid change would not be indistinguishable since we know that $f(y_1^0, y_2^0, z) \neq f(y_1^1, y_2^1, z)$ are not identical.

Solving this problem is the core technical aspect of our constructions and their analysis. At a very high level, we address this problem by introducing a new "flag" value that can change the nature of the function f that we are obfuscating to "disable" all plaintexts except for the ones that are in the challenge vectors. We provide more intuition in the full-versions.

Open Questions. Currently, our positive result for indistinguishability-based security requires that there to be a fixed polynomial limit on the size of challenge vectors, known at the time of setup. Unlike in the case of simulation security, we know of no corresponding lower bound showing that such a bound is necessary. Achieving full security without using complexity leveraging is another open question.

2 Multi-Input Functional Encryption

In this work, we study functional encryption for n-ary functions, where $n > 1$ (and in general, a polynomial in the security parameter). In other words, we are interested in encryption schemes where the owner of a "master" secret key can generate special keys SK_f that allow the computation of $f(x_1, \ldots, x_n)$ from n ciphertexts $\mathsf{CT}_1, \ldots, \mathsf{CT}_n$ corresponding to messages $x_1, \ldots, x_n$, respectively. We refer to such an encryption scheme as *multi-input* functional encryption. Analogously, we will refer to the existing notion of functional encryption (that only considers single-ary functions) as *single-input* functional encryption.

Intuitively, while single-input functional encryption can be viewed as a specific (non-interactive) way of performing two-party computation, our setting of multi-input functional encryption captures *multiparty* computation. Going forward with this analogy, we are interested in modeling the general scenario where the n input ciphertexts are computed by n *different* parties. This raises the following two important questions:

1. Do the parties (i.e., the encryptors) share the *same* encryption key or do they use *different* encryption keys EK_i to compute input ciphertexts CT_i.
2. Are the encryption keys secret or public?

As we shall see, these questions have important bearing on the security guarantees that can be achieved for multi-input functional encryption.

Towards that end, we present a general, unified syntax and security definitions for multi-input functional encryption. We consider encryption systems with n encryption keys, some of which may be public, while the rest are secret. When all of the encryption keys are public, then this represents the "public-key" setting, while when all the encryption keys are secret, then this represents the "secret-key" setting. Looking ahead, we remark that our modeling allows us to capture the intermediary cases between these two extremes that are interesting from the viewpoint of the security guarantees possible.

The rest of this section is organized as follows. We first present the syntax and correctness requirements for multi-input FE in Section 2.1). Then, in Section 2.2, we present our security definitions for multi-input FE. In Section 2.3 we give a construction that meets these definitions.

2.1 Syntax

Throughout the paper, we denote the security parameter by k. Let $\mathcal{X} = \{\mathcal{X}_k\}_{k\in\mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_k\}_{k\in\mathbb{N}}$ be ensembles where each $\mathcal{X}_k$ and $\mathcal{Y}_k$ is a finite set. Let $\mathcal{F} = \{\mathcal{F}_k\}_{k\in\mathbb{N}}$ be an ensemble where each $\mathcal{F}_k$ is a finite collection of n-ary functions. Each function $f \in \mathcal{F}_k$ takes as input n strings $x_1, \ldots, x_n$, where each $x_i \in \mathcal{X}_k$ and outputs $f(x_1, \ldots, x_n) \in \mathcal{Y}_k$.

A multi-input functional encryption scheme $\mathcal{FE}$ for $\mathcal{F}$ consists of four algorithms ($\mathsf{FE.Setup}$, $\mathsf{FE.Enc}$, $\mathsf{FE.Keygen}$, $\mathsf{FE.Dec}$) described below.

- **Setup** $\mathsf{FE.Setup}(1^k, n)$ is a PPT algorithm that takes as input the security parameter k and the function arity n. It outputs n encryption keys $\mathsf{EK}_1, \ldots, \mathsf{EK}_n$ and a master secret key MSK.
- **Encryption** $\mathsf{FE.Enc}(\mathsf{EK}, x)$ is a PPT algorithm that takes as input an encryption key $\mathsf{EK}_i \in (\mathsf{EK}_1, \ldots, \mathsf{EK}_n)$ and an input message $x \in \mathcal{X}_k$ and outputs a ciphertext CT.
  In the case where all of the encryption keys EK_i are the same, we assume that each ciphertext CT has an associated label i to denote that the encrypted plaintext constitutes an i'th input to a function $f \in \mathcal{F}_k$. For convenience of notation, we omit the labels from the explicit description of the ciphertexts. In particular, note that when EK_i's are *distinct*, the index of the encryption key EK_i used to compute CT implicitly denotes that the plaintext encrypted in CT constitutes an i'th input to f, and thus no explicit label is necessary.
- **Key Generation** $\mathsf{FE.Keygen}(\mathsf{MSK}, f)$ is a PPT algorithm that takes as input the master secret key MSK and an n-ary function $f \in \mathcal{F}_k$ and outputs a corresponding secret key SK_f.
- **Decryption** $\mathsf{FE.Dec}(\mathsf{SK}_f, \mathsf{CT}_1, \ldots, \mathsf{CT}_n)$ is a deterministic algorithm that takes as input a secret key SK_f and n ciphertexts $\mathsf{CT}_i, \ldots, \mathsf{CT}_n$ and outputs a string $y \in \mathcal{Y}_k$.

Definition 1 (Correctness). *A multi-input functional encryption scheme $\mathcal{FE}$ for $\mathcal{F}$ is* correct *if for all $f \in \mathcal{F}_k$ and all $(x_1, \ldots, x_n) \in \mathcal{X}_k^n$:*

$$\Pr\left[\begin{array}{l}(\mathsf{EK}, \mathsf{MSK}) \leftarrow \mathsf{FE.Setup}(1^k)\ ;\ \mathsf{SK}_f \leftarrow \mathsf{FE.Keygen}(\mathsf{MSK}, f)\ ; \\ \quad \mathsf{FE.Dec}\left(\mathsf{SK}_f, \mathsf{FE.Enc}\left(\mathsf{EK}_1, x_1\right), \ldots, \mathsf{FE.Enc}\left(\mathsf{EK}_n, x_n\right)\right) \neq f(x_1, \ldots, x_n)\end{array}\right] = \mathsf{negl}(k)$$

where the probability is taken over the coins of FE.Setup, FE.Keygen *and* FE.Enc.

2.2 Security for Multi-Input Functional Encryption

We now present our security definitions for multi-input functional encryption. We provide the most general possible definition with respect to adversarial corruptions, allowing the adversary to choose which subset of parties he corrupts, and which evaluation keys he will learn as a consequence. In Section 3 we will consider more restricted definitions.

Following the literature on single-input FE, we consider two notions of security, namely, indistinguishability-based security (or IND-security, in short) and simulation-based security (or SIM-security, in short).

Notation. We start by introducing some notation that is used in our security definitions. Let N denote the set of positive integers $\{1, \ldots, n\}$ where n denotes the arity of functions. For any two sets $S = \{s_0, \ldots, s_{|S|}\}$ and $I = \{i_1, \ldots, i_{|I|}\}$ such that $|I| \leq |S|$, we let S_I denote the subset $\{s_i\}_{i \in I}$ of the set S. Throughout the text, we use the vector and set notation interchangeably, as per convenience. For simplicity of notation, we omit explicit reference to auxiliary input to the adversary from our definitions.

Indistinguishability-Based Security. Here we present an indistinguishability-based security definition for multi-input FE.

Intuition. We start by giving an overview of the main ideas behind our indistinguishability-based security definition. To convey the core ideas, it suffices to consider the case of 2-ary functions. We will assume familiarity with the security definitions for single-input FE.

Let us start by considering the natural extension of *public-key* single-input FE to the two-input setting. That is, suppose there are two public encryption keys EK_1, EK_2 that are used to create ciphertexts of first inputs and second inputs, respectively, for 2-ary functions. Let us investigate what security can be achieved for *one* pair of challenge message tuples (x_1^0, x_2^0), (x_1^1, x_2^1) for the simplified case where the adversary makes secret key queries after receiving the challenge ciphertexts.

Suppose that the adversary queries secret keys for functions $\{f\}$. Now, recall that the IND-security definition in the single-input case guarantees that an adversary cannot differentiate between encryptions of x^0 and x^1 as long as $f(x^0) = f(x^1)$ for every $f \in \{f\}$. We note, however, that an analogous security guarantee cannot be achieved in the multi-input setting. That is, restricting

the functions $\{f\}$ to be such that $f(x_1^0, x_2^0) = f(x_1^1, x_2^1)$ is *not* enough since an adversary who knows both the encryption keys can create its own ciphertexts w.r.t. each encryption key. Then, by using the secret key corresponding to function f, it can learn additional values $\{f(x_1^b, \cdot)\}$ and $\{f(\cdot, x_2^b)\}$, where b is the challenge bit. In particular, if, for example, there exists an input x^* such that $f(x_1^0, x^*) \neq f(x_1^1, x^*)$, then the adversary can learn the challenge bit b! Therefore, we must enforce additional restrictions on the query functions f. Specifically, we must require that $f(x_1^0, x') = f(x_1^1, x')$ for *every* input x' in the domain (and similarly $f(x', x_2^0) = f(x', x_2^1)$). Note that this restriction "grows" with the arity n of the functions.

Let us now consider the secret-key case, where all the encryption keys are secret. In this case, for the above example, it suffices to require that $f(x_1^0, x_2^0) = f(x_1^1, x_2^1)$ since the adversary cannot create its own ciphertexts. Observe, however, that when there are *multiple* challenge messages, then an adversary can learn function evaluations over different "combinations" of challenge messages. In particular, if there are q challenge messages per encryption key, then the adversary can learn q^2 output values for every f. Then, we must enforce that for every $i \in [q^2]$, the i'th output value y_i^0 when challenge bit $b = 0$ is *equal* to the output value y_i^1 when the challenge bit $b = 1$.

The security guarantees in the public-key and the secret-key settings as discussed above are vastly different. In general, we observe that the *more* the number of encryption keys that are public, the *smaller* the class of functions that can be supported by the definition. Bellow, we present a unified definition that simultaneously captures the extreme cases of public-key and secret-key settings as well as all the "in between" cases.

Notation. Our security definition is parameterized by two variables t and q, where t denotes the number of encryption keys known to the adversary, and q denotes the number of challenge messages per encryption key. Thus, in total, the adversary is allowed to make $\mathsf{Q} = q \cdot n$ number of challenge message queries.

To facilitate the presentation of our IND security definition, we first introduce the following two notions:

Definition 2 (Function Compatibility). *Let $\{f\}$ be any set of functions $f \in \mathcal{F}_k$. Let $\mathsf{N} = \{1, \ldots, n\}$ and $\mathrm{I} \subseteq \mathsf{N}$. Then, a pair of message vectors $\boldsymbol{X}^0$ and $\boldsymbol{X}^1$, where $\boldsymbol{X}^b = \left\{x_{1,j}^b, \ldots, x_{n,j}^b\right\}_{j=1}^{q}$, are said to be* I-compatible *with $\{f\}$ if they satisfy the following property:*

- *For every $f \in \{f\}$, every $\mathrm{I}' = \{i_1, \ldots, i_t\} \subseteq \mathrm{I} \cup \emptyset$, every $j_1, \ldots, j_{n-t} \in [q]$, and every $x'_{i_1}, \ldots, x'_{i_t} \in \mathcal{X}_k$,*

$$f\left(\left\langle x_{i_1,j_1}^0, \ldots, x_{i_{n-t'},j_{n-t}}^0, x'_{i_1}, \ldots, x'_{i_t}\right\rangle\right) = f\left(\langle x_{i_1,j_1}^1, \ldots, x_{i_{n-t},j_{n-t}}^1, x'_{i_1}, \ldots, x'_{i_t}\rangle\right),$$

where $\langle y_{i_1}, \ldots, y_{i_n}\rangle$ denotes a permutation of the values $y_{i_1}, \ldots, y_{i_n}$ such that the value y_{i_j} is mapped to the ℓ'th location if y_{i_j} is the ℓ'th input (out of n inputs) to f.

Definition 3 (Message Compatibility). *Let* $\boldsymbol{X}^0$ *and* $\boldsymbol{X}^1$ *be any pair of message vectors , where* $\boldsymbol{X}^b = \left\{x^b_{1,j}, \ldots, x^b_{n,j}\right\}_{j=1}^{q}$. *Let* $\mathsf{N} = \{1, \ldots, k\}$ *and* $\mathrm{I} \subseteq \mathsf{N}$. *Then, a function* $f \in \mathcal{F}_k$ *is said to be* I-compatible *with* $(\boldsymbol{X}^0, \boldsymbol{X}^1)$ *if it satisfies the following property:*

- *For every* $\mathrm{I}' = \{i_1, \ldots, i_t\} \subseteq \mathrm{I} \cup \emptyset$, *every* $j_1, \ldots, j_{n-t} \in [q]$ *and every* $x'_{i_1}, \ldots, x'_{i_t} \in \mathcal{X}_k$,

$$f\left(\langle x^0_{i_1,j_1}, \ldots, x^0_{i_{n-t},j_{n-t}}, x'_{i_1}, \ldots, x'_{i_t}\rangle\right) = f\left(\langle x^1_{i_1,j_1}, \ldots, x^1_{i_{n-t},j_{n-t}}, x'_{i_1}, \ldots, x'_{i_t}\rangle\right),$$

We are now ready to present our formal definition for (t, q)-IND-secure multi-input functional encryption.

Definition 4 (Indistinguishability-based security). *We say that a multi-input functional encryption scheme* $\mathcal{FE}$ *for for* n*-ary functions* $\mathcal{F}$ *is* (t, q)-IND-secure *if for every PPT adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$, *the advantage of* $\mathcal{A}$ *defined as*

$$\mathsf{Adv}^{\mathcal{FE},\mathsf{IND}}_{\mathcal{A}}(1^k) = \left|\Pr[\mathsf{IND}^{\mathcal{FE}}_{\mathcal{A}}(1^k) = 1] - \frac{1}{2}\right|$$

is $\mathsf{negl}(k)$, *where:*

Experiment $\mathsf{IND}^{\mathcal{FE}}_{\mathcal{A}}(1^k)$*:*
$(\mathrm{I}, \mathsf{st}_0) \leftarrow \mathcal{A}_0(1^k)$ *where* $|\mathrm{I}| = t$
$(\mathbf{EK}, \mathsf{MSK}) \leftarrow \mathsf{FE.Setup}(1^k)$
$(\boldsymbol{X}^0, \boldsymbol{X}^1, \mathsf{st}_1) \leftarrow \mathcal{A}_1^{\mathsf{FE.Keygen}(\mathsf{MSK},\cdot)}(\mathsf{st}_0, \mathbf{EK}_{\mathrm{I}})$ *where* $\boldsymbol{X}^\ell = \left\{x^\ell_{1,j}, \ldots, x^\ell_{n,j}\right\}_{j=1}^{q}$
$b \leftarrow \{0,1\}$; $\mathsf{CT}_{i,j} \leftarrow \mathsf{FE.Enc}(\mathsf{EK}_i, x^b_{i,j})$ $\forall i \in [n]$, $j \in [q]$
$b' \leftarrow \mathcal{A}_2^{\mathsf{FE.Keygen}(\mathsf{MSK},\cdot)}(\mathsf{st}_1, \mathbf{CT})$
Output: $(b = b')$

In the above experiment, we require:

- **Compatibility with Function Queries:** *Let* $\{f\}$ *denote the entire set of key queries made by* $\mathcal{A}_1$. *Then, the challenge message vectors* $\boldsymbol{X}_0$ *and* $\boldsymbol{X}_1$ *chosen by* $\mathcal{A}_1$ *must be* I*-compatible with* $\{f\}$.
- **Compatibility with Ciphertext Queries:** *Every key query* g *made by* $\mathcal{A}_2$ *must be* I*-compatible with* $\boldsymbol{X}^0$ *and* $\boldsymbol{X}^1$.

Selective Security. We also consider selective *indistinguishability*-based security for multi-input functional encryption. Formally, (t, q)-sel-IND-security is defined in the same manner as Definition 4, except that the adversary $\mathcal{A}_1$ is required to choose the challenge message vectors $\boldsymbol{X}^0$, $\boldsymbol{X}^1$ *before* the evaluation keys $\mathbf{EK}$ and

the master secret key MSK are chosen by the challenger. We omit the formal definition to avoid repetition.

Simulation-Based Security. Here we present a simulation-based security definition for multi-input FE. We consider the case where the adversary makes key queries *after* choosing the challenge messages. That is, we only consider *adaptive* key queries.

Our definition extends the simulation-based security definition for single-input FE that supports adaptive key queries[6,16,3,8]. In particular, we present a general definition that models both black-box and non-black-box simulation.

Intuition. We start by giving an overview of the main ideas behind our simulation-based security definition. To convey the core ideas, it suffices to consider the case of 2-ary functions. Let us start by considering the natural extension of *public-key* single-input FE to the two-input setting. That is, suppose there are two public encryption keys EK_1, EK_2 that are used to create ciphertexts of first inputs and second inputs, respectively, for 2-ary functions. Let us investigate what security can be achieved for *one* challenge message tuple (x_1, x_2).

Suppose that the adversary queries secret keys for functions $\{f\}$. Now, recall that the SIM-security definition in the single-input case guarantees that for every $f \in \{f\}$, an adversary cannot learn more than $f(x)$ when x is the challenge message. We note, however, that an analogous security guarantee cannot be achieved in the multi-input setting. Indeed, an adversary who knows both the encryption keys can create its own ciphertexts w.r.t. each encryption key. Then, by using the secret key corresponding to function f, it can learn additional values $\{f(x_1, \cdot)\}$ and $\{f(\cdot, x_2)\}$. Thus, we must allow for the ideal world adversary, aka simulator, to learn the same information.

In the secret-key case, however, since all of the encryption keys are secret, the SIM-security definition for single-input FE indeed extends in a natural manner to the multi-input setting. We stress, however, that when there are multiple challenge messages, we must take into account the fact that adversary can learn function evaluations over all possible "combinations" of challenge messages. Our definition presented below formalizes this intuition.

Similar to the IND-security case, our definition is parameterized by variables t and q as defined earlier. We now formally define (t, q)-SIM-secure multi-input functional encryption.

Definition 5 (Simulation-based Security). *We say that a functional encryption scheme $\mathcal{FE}$ for n-ary functions $\mathcal{F}$ is* (t, q)-SIM-secure *if for every PPT adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$, there exists a PPT simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2)$ such that the outputs of the following two experiments are computationally indistinguishable:*

Experiment $\mathsf{REAL}_{\mathcal{A}}^{\mathcal{FE}}(1^k)$:
$(\mathrm{I}, \mathsf{st}_0) \leftarrow \mathcal{A}_0(1^k)$ *where* $|\mathrm{I}| = t$
$(\mathbf{EK}, \mathsf{MSK}) \leftarrow \mathsf{FE.Setup}(1^k)$
$(\mathcal{M}, \mathsf{st}_1) \leftarrow \mathcal{A}_1(\mathsf{st}_0, \mathbf{EK}_{\mathrm{I}})$
$\boldsymbol{X} \leftarrow \mathcal{M}$ *where* $\boldsymbol{X} = \{x_{1,j}, \ldots, x_{n,j}\}_{j=1}^{q}$
$\mathsf{CT}_{i,j} \leftarrow \mathsf{FE.Enc}(\mathsf{EK}_i, x_{i,j})\ \forall i \in [n],\ j \in [q]$
$\alpha \leftarrow \mathcal{A}_2^{\mathsf{FE.Keygen}(\mathsf{MSK},\cdot)}(\mathbf{CT}, \mathsf{st}_1)$
Output: $(\mathrm{I}, \mathcal{M}, \boldsymbol{X}, \{f\}, \alpha)$

Experiment $\mathsf{IDEAL}_{\mathcal{S}}^{\mathcal{FE}}(1^k)$:
$(\mathrm{I}, \mathsf{st}_0) \leftarrow \mathcal{S}_0(1^k)$
$(\mathcal{M}, \mathsf{st}_1) \leftarrow \mathcal{S}_1(\mathsf{st}_0)$
$\alpha \leftarrow \mathcal{S}_2^{\mathsf{TP}(\mathcal{M},\cdot,\cdot)}(\mathsf{st}_1)$
Output: $(\mathrm{I}, \mathcal{M}, \boldsymbol{X}, \{g\}, \alpha)$

where the oracle $\mathsf{TP}(\mathcal{M}, \cdot, \cdot)$ *denotes the ideal world trusted party,* $\{f\}$ *denotes the set of queries of* $\mathcal{A}_2$ *to* $\mathsf{FE.Keygen}$ *and* $\{g\}$ *denotes the set of functions appearing in the queries of* $\mathcal{S}_2$ *to* TP. *Given the message distribution* $\mathcal{M}$, TP *first samples a message vector* $\boldsymbol{X} \leftarrow M$, *where* $\boldsymbol{X} = \{x_{1,j}, \ldots, x_{n,j}\}_{j=1}^{q}$. *It then accepts queries of the form* $\left(g, (j_1, \ldots, j_{n-p}), \left(x'_{i'_1}, \ldots, x'_{i'_p}\right)\right)$ *where* $p \leq t$, $\{i'_1, \ldots, i'_p\} \subseteq \mathrm{I} \cup \emptyset$ *and* $x'_{i'_1}, \ldots, x'_{i'_p} \in \mathcal{X}_k$. *On receiving such a query,* TP *outputs:*

$$g\left(\left\langle x_{i_1,j_1}, \ldots, x_{i_{n-p},j_{n-p}}, x'_{i'_1}, \ldots, x'_{i'_p}\right\rangle\right),$$

where $\langle y_{i_1}, \ldots, y_{i_n}\rangle$ *denotes a permutation of the values* $y_{i_1}, \ldots, y_{i_n}$ *such that the value* y_{i_j} *is mapped to the* ℓ*'th location if* y_{i_j} *is the* ℓ*'th input (out of* n *inputs) to* g.

Remark 1 (On Queries to the Trusted Party). Note that when $t = 0$, then given the challenge ciphertexts $\mathbf{CT}$, intuitively, the real adversary can only compute values $\mathsf{FE.Dec}(\mathsf{SK}_f, \mathsf{CT}_{1,j_1}, \ldots, \mathsf{CT}_{n,j_n})$ for every $j_i \in [q]$, $i \in [n]$. To formalize the intuition that this adversary does not learn anything more than function values $\{f(x_{1,j_1}, \ldots, x_{n,j_n})\}$, we restrict the ideal adversary aka simulator to learn exactly this information.

However, when $t > 0$, then the real adversary can compute values:

$$\mathsf{FE.Dec}\left(\mathsf{SK}_f, \left\langle \mathsf{CT}_{i_1,j_1}, \ldots, \mathsf{CT}_{i_{n-t},j_{n-t}}, \mathsf{CT}'_{i'_1}, \ldots, \mathsf{CT}'_{i'_t}\right\rangle\right)$$

for ciphertexts $\mathsf{CT}'_{i'_\ell}$ of its choice since it knows the encryption keys $\mathbf{EK}_{\mathrm{I}}$. In other words, such an adversary can learn function values of the form $f\left(\left\langle x_{i_1,j_1}, \ldots, x_{i_{n-t},j_{n-t}}, \cdot, \ldots, \cdot\right\rangle\right)$. Thus, we must provide the same ability to the simulator as well. Our definition presented above precisely captures this.

Selective Security. We also consider *selective* simulation-based security for multi-input functional encryption. Formally, (t, q)-sel-SIM-security is defined in the same manner as Definition 5, except that in the real world experiment, adversary $\mathcal{A}_1$ chooses the message distribution $\mathcal{M}$ *before* the evaluation keys $\mathbf{EK}$ and the master secret key MSK are chosen by the challenger. We omit the formal definition to avoid repetition.

*Remark 2 (*SIM*-security: Secret-key setting).* When $t = 0$, *none* of the encryption keys are known to the adversary. In this "secret-key" setting, there is no difference between $(0, q)$-sel-SIM-security and $(0, q)$-SIM-security.

Impossibility of $(0, \mathsf{poly}(k))$-SIM*-security.* We note that the lower bounds of [6,3] already establish that it is impossible to achieve $(0, \mathsf{poly}(k))$-SIM-secure functional encryption for 1-ary functions. In particular, [6] prove their result for the IBE functionality, while the [3] impossibility result is given for almost all 1-ary functionalities (assuming the existence of collision-resistance hash functions). The positive results in this paper for SIM-secure multi-input FE are consistent with these negative results. That is, our constructions (for general functionalities) provide SIM security only for the case where the number of challenge messages q are a priori bounded.

2.3 A Construction for the General Case

Let $\mathcal{F}$ denote the family of all efficiently computable (deterministic) n-ary functions. We now present a functional encryption scheme $\mathcal{FE}$ for $\mathcal{F}$. Assuming the existence of one-way functions and indistinguishability obfuscation for all efficiently computable circuits, we prove the following security guarantees for $\mathcal{FE}$:

1. For $t = 0$, and any $q = q(k)$ such that $\binom{qn}{n} = \mathsf{poly}(k)$, $\mathcal{FE}$ is $(0, q)$-SIM-secure.[1] In this case, the size of the secret keys in $\mathcal{FE}$ grows linearly with $\binom{qn}{n}$.
2. For any $t \leq n$ and $q = \mathsf{poly}(k)$, $\mathcal{FE}$ is (t, q)-sel-IND-secure. In this case, the size of the secret keys is independent of q.

Note that by using standard complexity leveraging, we can extend the second result to show that $\mathcal{FE}$ is, in fact, (t, q)-IND-secure. Note that in this case, we would require the indistinguishability obfuscator $i\mathcal{O}$ (and the one-way function) to be secure against adversaries running in time $\mathcal{O}(2^M)$, where M denotes the total length of the challenge message vectors.

Notation. Let $(\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Verify})$ be a NIWI proof system. Let Com denote a perfectly binding commitment scheme. Let $i\mathcal{O}$ denote an indistinguishability obfuscator. Finally, let $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ be a semantically secure public-key encryption scheme. We denote the length of ciphertexts in PKE by $\mathsf{c\text{-}len} = \mathsf{c\text{-}len}(k)$. Let $\mathsf{len} = 2 \cdot \mathsf{c\text{-}len}$.

We now proceed to describe our scheme $\mathcal{FE} = (\mathsf{FE.Setup}, \mathsf{FE.Enc}, \mathsf{FE.Keygen}, \mathsf{FE.Dec})$.

Setup $\mathsf{FE.Setup}(1^k)$*:* The setup algorithm first computes a CRS $\mathsf{crs} \leftarrow \mathsf{CRSGen}(1^k)$ for the NIWI proof system. Next, it computes two key pairs – $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow$

[1] Recall that when $t = 0$, there is no difference between selective security and standard security as defined in Section 2.2. See Remark 2.

$\mathsf{PKE.Setup}(1^k)$ and $(\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow \mathsf{PKE.Setup}(1^k)$ – of the public-key encryption scheme PKE. Finally, it computes the following commitments: (a) $Z_1^{i,j} \leftarrow \mathsf{Com}(0^{\mathsf{len}})$ for every $i \in [n]$, $j \in [q]$. (b) $Z_2^i \leftarrow \mathsf{Com}(0)$ for every $i \in [n]$.

For every $i \in [n]$, the i'th encryption key $\mathsf{EK}_i = \left(\mathsf{crs}, \mathsf{pk}_1, \mathsf{pk}_2, \left\{Z_1^{i,j}\right\}, Z_2^i, r_2^i\right)$ where r_2^i is the randomness used to compute the commitment Z_2^i. The master secret key is set to be $\mathsf{MSK} = \left(\mathsf{crs}, \mathsf{pk}_1, \mathsf{pk}_2, \mathsf{sk}_1, \left\{Z_1^{i,j}\right\}, \left\{Z_2^i\right\}\right)$. The setup algorithm outputs $(\mathsf{EK}_1, \ldots, \mathsf{EK}_n, \mathsf{MSK})$.

Encryption $\mathsf{FE.Enc}(\mathsf{EK}_i, x)$*:* To encrypt a message x with the i'th encryption key EK_i, the encryption algorithm first computes $c_1 \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_1, x)$ and $c_2 \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_2, x)$. Next, it computes a NIWI proof $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, y, w)$ for the statement $y = \left(c_1, c_2, \mathsf{pk}_1, \mathsf{pk}_2, \left\{Z_1^{i,j}\right\}, Z_2^i\right)$:

- *Either* c_1 and c_2 are encryptions of the same message and Z_2^i is a commitment to 0, *or*
- $\exists\ j \in [q]$ s.t. $Z_1^{i,j}$ is a commitment to $c_1 \| c_2$.

A witness $w_{\mathsf{real}} = (m, s_1, s_2, r_2^i)$ for the first part of the statement, referred to as the *real witness*, includes the message m and the randomness s_1 and s_2 used to compute the ciphertexts c_1 and c_2, respectively, and the randomness r_2^i used to compute Z_2^i. A witness $w_{\mathsf{trap}} = (j, r_1^{i,j})$ for the second part of the statement, referred to as the *trapdoor witness*, includes an index j and the randomness $r_1^{i,j}$ used to compute $Z_1^{i,j}$.

The honest encryption algorithm uses the real witness w_{real} to compute π. The output of the algorithm is the ciphertext $\mathsf{CT} = (c_1, c_2, \pi)$.

Key Generation $\mathsf{FE.Keygen}(\mathsf{MSK}, f)$*:* The key generation algorithm on input f computes $\mathsf{SK}_f \leftarrow i\mathcal{O}(\mathsf{G}_f)$ where the function G_f is defined in Figure 1. Note that G_f has the master secret key MSK hardwired in its description.

$\mathsf{G}_f(\mathsf{CT}_1, \ldots, \mathsf{CT}_n)$

1. For every $i \in [n]$:
   (a) Parse $\mathsf{CT}_i = (c_{i,1}, c_{i,2}, \pi_i)$.
   (b) Let $y_i = \left(c_{i,1}, c_{i,2}, \mathsf{pk}_1, \mathsf{pk}_2, \left\{Z_1^{i,j}\right\}, Z_2^i\right)$ be the statement corresponding to the proof string π_i. If $\mathsf{Verify}(\mathsf{crs}, y_i, \pi_i) = 0$, then stop and output $\perp$. Otherwise, continue to the next step.
   (c) Compute $x_i \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}_1, c_{i,1})$.
2. Output $f(x_1, \ldots, x_n)$.

Fig. 1. Functionality G_f

The algorithm outputs SK_f as the secret key for f.

Size of Function G_f. In order to prove that $\mathcal{FE}$ is $(0,q)$-SIM-secure, we require the function G_f to be padded with zeros such that $|\mathsf{G}_f| = |\mathsf{Sim}.\mathsf{G}_f|$, where the "simulated" functionality $\mathsf{Sim}.\mathsf{G}_f$ is described in the full version. In this case, the size of SK_f grows linearly with $\binom{qn}{n}$.

Note, however, that such a padding is *not* necessary to prove (t,q)-sel-IND-security for $\mathcal{FE}$. Indeed, in this case, the secret keys SK_f are independent of the number of message queries q made by the adversary.

Decryption $\mathsf{FE.Dec}(\mathsf{SK}_f, \mathsf{CT}_1, \ldots, \mathsf{CT}_n)$*:* The decryption algorithm on input $(\mathsf{CT}_1, \ldots, \mathsf{CT}_n)$ computes and outputs $\mathsf{SK}_f(\mathsf{CT}_1, \ldots, \mathsf{CT}_n)$.

This completes the description of our functional encryption scheme $\mathcal{FE}$. The correctness property of the scheme follows from inspection. In the full version by Goldwasser et al. [10], we prove that $\mathcal{FE}$ is $(0,q)$-SIM-secure, and that $\mathcal{FE}$ is (t,q)-sel-IND-secure (and (t,q)-IND-secure via complexity leveraging).

3 Restricted Security Notions

In this section we present indistinguishability-based definitions and constructions for two restricted settings: the symmetric key setting in which the adversary does not learn any evaluation keys, and a setting in which all clients operate in fixed *time-steps*, encrypting only one plaintext in each time-step. In this latter setting, we do not allow functions to compute on ciphertexts from different time-steps. A full exposition appears in the full version by Gordon et al. [12].

3.1 The Symmetric-Key Setting

For simplicity, we focus on the binary-input setting in the symmetric key setting. However, we remark that our construction extends naturally to the n-ary 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.

Definitions. Let $\mathcal{F} = \{\mathcal{F}_n\}_{n>0}$ be a collection of function families, where every $f \in \mathcal{F}_n$ is a polynomial time function $f : \{0,1\}^{m_1(n)} \times \{0,1\}^{m_2(n)} \to \Sigma$. A binary symmetric key FE scheme supporting $\mathcal{F}$ is a collection of 4 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 $f \in \mathcal{F}_n$, $\mathsf{TK}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$

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

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

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

We now define security for IND-secure symmetric-key binary FE. In the full version by Gordon et al.[12] we provide the stronger, adaptive security definition. Unlike in the public key setting, here single-message security does not imply multi-message security, so we cannot prove a parallel to Lemma 1. (Technically, the problem arises in the reduction, where the simulator cannot create the necessary ciphertexts for the hybrid world without knowing the secret key.) Instead, we only define the multi-message variant.

Our construction below only achieves selective security, but we note that we can achieve adaptive security through standard complexity-leveraging techniques. (We omit the details.)

Selective security. 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. $\{(x_1, \ldots, x_n), (y_1, \ldots, y_n)\} \leftarrow \mathcal{A}(1^\kappa)$
2. $(\mathsf{msk}, \mathsf{param}) \leftarrow \mathsf{Setup}(1^\kappa)$
3. $b \leftarrow \{0, 1\}$
4. if $b = 0$: $\forall i \in [n] : \mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{msk}, x_i)$, else: $\forall i \in [n] : \mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{msk}, y_i)$.
5. $b' \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\cdot)}(\mathsf{param}, \mathsf{CT}_1, \ldots, \mathsf{CT}_n)$

An adversary is considered *non-trivial* if for every query f made to the $\mathsf{KeyGen}(\cdot)$ oracle, and for all $i, j \in [n]$, 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.

A Construction

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 the full version [12]. 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. $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\kappa)$
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 $\mathsf{param} := (\mathsf{crs}, \mathsf{pk}, \mathsf{pk}', \mathsf{com})$, $\mathsf{msk} := (\mathsf{sk}, \mathsf{sk}', \alpha, r)$

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 $f(x_0, x_1)$.

Fig. 2. Symmetric-key IND-secure binary FE: Program P

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

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.

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

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

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

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

In the full version we provide a proof of the following theorem [12].

Theorem 1. *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 3.1*

Instantiation and efficiency. If we use the approach described in our full version for constructing the SSS-NIZK, the ciphertext is succinct, and is $\mathsf{poly}(\kappa)$ in size [12]. For a scheme tolerant up to n ciphertext queries, the public parameter size, encryption time, decryption time are $O(n)\mathsf{poly}(\kappa)$. The reason for the dependence on n is due to the simulator's need to simultaneously simulate $O(n)$ SSS-NIZKs in the simulation, which increases the size of the crs. Removing the dependence on n remains an important open problem.

3.2 Time-dependent Setting

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

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

$\mathsf{Enc} : \mathsf{CT} \leftarrow \mathsf{Enc}(\mathsf{usk}_i, x, t)$, here $t \in \mathbb{N}$ denotes the current time step

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

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

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 $\mathsf{Dec}(\mathsf{KeyGen}(\mathsf{msk}, f), \mathsf{Enc}(\mathsf{usk}_1, x_1, t), \ldots, \mathsf{Enc}(\mathsf{usk}_n, x_n, t)) = f(x_1, x_2, \ldots, x_n)$.

Below we define a selectively secure indistinguishability-based security for binary FE. In the full version [12] we provide two other, stronger security definitions, both allowing adaptive plaintext challenges. There we prove the following Lemma stating that the two notions are equivalent.

Lemma 1. *Adaptive, multi-message indistinguishability security is equivalent to adaptive, single-message indistinguishability security.*

Our construction below only achieves selective security, but we note that we can achieve the stronger definitions through standard complexity-leveraging techniques [Folklore]. (We omit the details.)

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 $\boldsymbol{x}$ to denote a vector $\boldsymbol{x} := (x_1, x_2, \ldots, x_n)$. 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 $\mathbf{CT}_G \leftarrow \mathsf{Enc}(\mathbf{usk}_G, \boldsymbol{x}_G, t)$ to denote the following: $\forall i \in G : \mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{usk}_i, x_i, t)$. We use the shorthand $f(\boldsymbol{x}_G, \cdot) : \mathcal{D}^{|\overline{G}|} \to \Sigma$ to denote a function restricted to a subset G on inputs denoted $\boldsymbol{x}_G \in \mathcal{D}^{|G|}$. Let $h := f(\boldsymbol{x}_G, \cdot)$, then by our definition, $h(\boldsymbol{x}_{\overline{G}}) := f(\boldsymbol{x})$.

Selective security. We define a relaxation of the above security notion called selective security. 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 $E_G(\boldsymbol{x}_G)$, the oracle increments the current time step $t \leftarrow t + 1$, and returns $\mathsf{Enc}(\mathbf{usk}_G, \boldsymbol{x}_G, t)$.

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

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

Definition 6 (Selective IND-security of MC-FE). *We say that an MC-FE scheme is selectively and indistinguishably secure, if for any polynomial-time, non-trivial adversary $\mathcal{A}$ in the above selective security game,* $\left|\Pr[b' = b] - \frac{1}{2}\right| \leq \mathsf{negl}(\kappa)$.

A Construction Intuition: In this setting, the adversary is allowed to corrupt some set $\bar{G}$. Our restriction on the adversary is that for challenge vectors $\boldsymbol{x}_G$ and $\boldsymbol{y}_G$, $f(\boldsymbol{x}_G, \cdots) = f(\boldsymbol{y}_G, \cdots)$, where $\boldsymbol{x}_G$ and $\boldsymbol{y}_G$ 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 $(\mathsf{Enc}(\boldsymbol{x}_G), \mathsf{Enc}(\boldsymbol{y}_G))$ 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 $\boldsymbol{\beta}_G$ such that $\langle \boldsymbol{\beta}_G, \boldsymbol{\alpha}_G \rangle = 0$. We hard-code $\boldsymbol{\beta}_G$ in the $i\mathcal{O}$, and if the $\boldsymbol{\alpha}_G$ values in the ciphertexts are not orthogonal to $\boldsymbol{\beta}_G$, the $i\mathcal{O}$ will simply output $\perp$.

In the hybrid world, instead of using the honest vector $\boldsymbol{\alpha}_G$, the simulator uses another random $\boldsymbol{\alpha}'_G$ orthogonal to $\boldsymbol{\beta}_G$, 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 $\boldsymbol{\alpha}_G$ and $\boldsymbol{\alpha}'_G$ will result in a vector *not* orthogonal to $\boldsymbol{\beta}_G$ (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 $\boldsymbol{\alpha}_G$ directly in the ciphertexts, we use a generator for a group that supports the Diffie-Hellman assumption and encode $g^{\boldsymbol{\alpha}_G}$ 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.

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:$\mathsf{CT} := (t, i, c, c', d, \pi)$.
  $\exists\ m, (\rho, \rho'), \omega$ s.t.

$$\mathsf{DH}(h_t, g_i, d, \omega) \wedge (c = \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, m; \rho)) \wedge \left(c' = \mathcal{E}.\mathsf{Enc}(\mathsf{pk}', m; \rho')\right)$$

  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:

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

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

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

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:

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

Fig. 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 $\mathsf{TK}_f(\mathsf{CT}_1, \mathsf{CT}_2, \ldots, \mathsf{CT}_n)$.

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

Internal hard-coded state: $\mathsf{param} = (\mathsf{crs}, \mathsf{pk}, \mathsf{pk}', g, \{g_i\}_{i\in[n]})$, 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 $f(x_1, x_2, \ldots, x_n)$.

Fig. 4. MC-FE: Program P

Theorem 2. *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 3.2.*

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 $O(n + T)\mathsf{poly}(\kappa)$ in size, where n denotes the number of parties, and T denotes an upper bound on the number of time steps.

Instantiation and efficiency. We can instantiate our scheme using the SSS-NIZK construction and the $i\mathcal{O}$ construction described by Garg et. al [9]. In this way, our ciphertext is succinct, and is only $\mathsf{poly}(\kappa)$ in size. Letting n denote the number of parties, the encryption time is $O(n)\mathsf{poly}(\kappa)$, and the decryption time is $O(n+|f|)\cdot\mathsf{poly}(\kappa)$. 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 $O(n)\mathsf{poly}(\kappa)$ for the version with the random oracle, and is $O(n+T)\mathsf{poly}(\kappa)$ 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.

References

1. Agrawal, S., Gorbunov, S., Vaikuntanathan, V., Wee, H.: Functional encryption: New perspectives and lower bounds. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 500–518. Springer, Heidelberg (2013)
2. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S.P., Yang, K.: On the (im)possibility of obfuscating programs. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 1–18. Springer, Heidelberg (2001)
3. Bellare, M., O'Neill, A.: Semantically-secure functional encryption: Possibility results, impossibility results and the quest for a general definition. In: Abdalla, M., Nita-Rotaru, C., Dahab, R. (eds.) CANS 2013. LNCS, vol. 8257, pp. 218–234. Springer, Heidelberg (2013)
4. Boldyreva, A., Chenette, N., Lee, Y., O'Neill, A.: Order-preserving symmetric encryption. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 224–241. Springer, Heidelberg (2009)
5. Boldyreva, A., Chenette, N., O'Neill, A.: Order-preserving encryption revisited: Improved security analysis and alternative solutions. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 578–595. Springer, Heidelberg (2011)
6. Boneh, D., Sahai, A., Waters, B.: Functional encryption: Definitions and challenges. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 253–273. Springer, Heidelberg (2011)
7. Boneh, D., Waters, B.: Conjunctive, subset, and range queries on encrypted data. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 535–554. Springer, Heidelberg (2007)
8. De Caro, A., Iovino, V., Jain, A., O'Neill, A., Paneth, O., Persiano, G.: On the achievability of simulation-based security for functional encryption. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 519–535. Springer, Heidelberg (2013)
9. Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. In: FOCS (2013), `http://eprint.iacr.org/2013/451`
10. Goldwasser, S., Goyal, V., Jain, A., Sahai, A.: Multi-input functional encryption. Cryptology ePrint Archive, Report 2013/727 (2013), `http://eprint.iacr.org/`
11. Goldwasser, S., Kalai, Y.T., Popa, R.A., Vaikuntanathan, V., Zeldovich, N.: Reusable garbled circuits and succinct functional encryption. In: STOC (2013)
12. Gordon, S.D., Katz, J., Liu, F.-H., Shi, E., Zhou, H.-S.: Multi-input functional encryption. Cryptology ePrint Archive, Report 2013/774 (2013), `http://eprint.iacr.org/`
13. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: ACM Conference on Computer and Communications Security (2006)
14. Katz, J., Sahai, A., Waters, B.: Predicate encryption supporting disjunctions, polynomial equations, and inner products. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 146–162. Springer, Heidelberg (2008)
15. Lewko, A., Okamoto, T., Sahai, A., Takashima, K., Waters, B.: Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 62–91. Springer, Heidelberg (2010)
16. O'Neill, A.: Definitional issues in functional encryption. IACR Cryptology ePrint Archive, 2010 (2010)

17. Pandey, O., Rouselakis, Y.: Property preserving symmetric encryption. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 375–391. Springer, Heidelberg (2012)
18. Sahai, A., Seyalioglu, H.: Worry-free encryption: functional encryption with public keys. In: ACM Conference on Computer and Communications Security, pp. 463–472 (2010)
19. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 457–473. Springer, Heidelberg (2005)
20. Sahai, A., Waters, B.: How to use indistinguishability obfuscation: Deniable encryption, and more. IACR Cryptology ePrint Archive, 2013 (2013)