Multi-Input Functional Encryption for Unbounded Arity Functions

Saikrishna Badrinarayanan*, Divya Gupta**,
Abhishek Jain***, and Amit Sahai†

Abstract. The notion of multi-input functional encryption (MI-FE) was recently introduced by Goldwasser et al. [EUROCRYPT'14] as a means to non-interactively compute aggregate information on the joint private data of multiple users. A fundamental limitation of their work, however, is that the total number of users (which corresponds to the arity of the functions supported by the MI-FE scheme) must be a priori *bounded* and fixed at the system setup time.

In this work, we overcome this limitation by introducing the notion of unbounded input MI-FE that supports the computation of functions with *unbounded arity*. We construct such an MI-FE scheme with indistinguishability security in the selective model based on the existence of public-coin differing-inputs obfuscation for turing machines and collision-resistant hash functions.

Our result enables several new exciting applications, including a new paradigm of *on-the-fly* secure multiparty computation where new users can join the system dynamically.

* University of California, Los Angeles and Center for Encrypted Functionalities. Email: saikrishna@cs.ucla.edu

** University of California, Los Angeles and Center for Encrypted Functionalities. Email: divyag@cs.ucla.edu

*** Johns Hopkins University and Center for Encrypted Functionalities. Email: abhishek@cs.jhu.edu Supported in part by a DARPA/ARL Safeware Grant W911NF-15-C-0213 and NSF CNS-1414023

† University of California, Los Angeles and Center for Encrypted Functionalities. Email: sahai@cs.ucla.edu Research supported in part from a DARPA/ONR PROCEED award, a DARPA/ARL SAFEWARE award, NSF Frontier Award 1413955, 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.

1 Introduction

Functional Encryption. Traditionally, encryption has been used as a tool for private end-to-end communication. The emergence of cloud computing has opened up a host of new application scenarios where more functionality is desired from encryption beyond the traditional privacy guarantees. To address this challenge, the notion of functional encryption (FE) has been developed in a long sequence of works [21,13,4,16,17,3,19]. In an FE scheme for a family $\mathcal{F}$, it is possible to derive decryption keys K_f for any function $f \in \mathcal{F}$ from a master secret key. Given such a key K_f and an encryption of a message x, a user can compute $f(x)$. Intuitively, the security of FE says that an adversarial user should only learn $f(x)$ and "nothing else about x."

Multi-Input Functional Encryption. Most of the prior work on FE focuses on the problem of computing a function over a *single* plaintext given its corresponding ciphertext. However, many applications require the computation of aggregate information from *multiple* data sources (that may correspond to different users). To address this issue, recently, Goldwasser et al. [10] introduced the notion of multi-input functional encryption (MI-FE). Let $\mathcal{F}$ be a family of n-ary functions where n is a polynomial in the security parameter. In an MI-FE scheme for $\mathcal{F}$, the owner of the master secret key (as in FE) can compute decryption keys K_f for any function $f \in \mathcal{F}$. The new feature in MI-FE is that K_f can be used to compute $f(x_1, \ldots, x_n)$ from n ciphertexts $\mathrm{CT}_1, \ldots, \mathrm{CT}_n$ of messages $x_1, \ldots, x_n$ respectively, where each CT_i is computed *independently*, possibly using a different encryption key (but w.r.t. the same master secret key).

As discussed in [10] (see also [11,12]), MI-FE enables several important applications such as computing on multiple encrypted databases, order-revealing and property-revealing encryption, multi-client delegation of computation, secure computation on the web [14] and so on. Furthermore, as shown in [10], MI-FE, in fact, implies program obfuscation [2,8].

A fundamental limitation of the work of Goldwasser et al [10] is that it requires an a priori (polynomial) bound on the arity n of the function family $\mathcal{F}$. More concretely, the arity n of the function family must be fixed during system setup when the parameters of the scheme are generated. This automatically fixes the number of users in the scheme and therefore new users cannot join the system at a later point of time. Furthermore, the size of the system parameters and the complexity of the algorithms depends on n. This has an immediate adverse impact on the applications of MI-FE: for example, if we use the scheme of [10] to compute on multiple encrypted databases, then we must a priori fix the number of databases and use decryption keys of size proportional to the number of databases.

Our Question: Unbounded Arity MI-FE. In this work, we seek to overcome this limitation. Specifically, we study the problem of MI-FE for general functions $\mathcal{F}$ with *unbounded* arity. Note that this means that the combined length of all the inputs to any function $f \in \mathcal{F}$ is unbounded and hence we must work in the

Turing machine model of computation (as opposed to circuits). In addition, we also allow for each individual input to f to be of unbounded length.

More concretely, we consider the setting where the owner of a master secret key can derive decryption keys K_M for a general Turing machine M. For any index $i \in 2^\lambda$ (where λ is the security parameter), the owner of the master secret key can (at any point in time) compute an encryption key EK_i. Finally, given a list of ciphertexts $\mathrm{CT}_1, \ldots, \mathrm{CT}_\ell$ for any arbitrary ℓ, where each CT_i is encryption of some message x_i w.r.t. EK_i, and a decryption key K_M, one should be able to learn $M(x_1, \ldots, x_\ell)$.

We formalize security via a natural generalization of the indistinguishability-based security framework for bounded arity MI-FE to the case of unbounded arity. We refer the reader to Section 3 for details but point out that similar to [10], we also focus on selective security where the adversary declares the challenge messages at the beginning of the game.

1.1 Our Results

Our main result is an MI-FE scheme for functions with unbounded arity assuming the existence of public-coin differing-inputs obfuscation (pc-diO) [20] for general Turing machines with unbounded input length and collision-resistant hash functions. We prove indistinguishability-based security of our scheme in the selective model.

Theorem 1 (Informal). *If public-coin differing-inputs obfuscation for general Turing machines and collision-resistant hash functions exist, then there exists an indistinguishably-secure MI-FE scheme for general functions with unbounded arity, in the selective model.*

Discussion. Recently, Pandey et al. [20] defined the notion of pc-diO as a weakening of differing-inputs obfuscation (diO) [2,5,1]. In the same work, they also give a construction of pc-diO for general Turing machines with unbounded input length based on pc-diO for general circuits and public-coin (weak) succinct non-interactive arguments of knowledge (SNARKs).[1] We note that while the existence of diO has recently come under scrutiny [9], no impossibility results are known for pc-diO.

On the Necessity of Obfuscation. It was shown by Goldwasser et al. [10] that MI-FE for bounded arity functions with indistinguishability-based security implies indistinguishability obfuscation for general circuits. A straightforward extension of their argument (in the case where at least one of the encryption keys is known to the adversary) shows that MI-FE for functions with unbounded arity

[1] A recent work by [6] shows that SNARKs with privately generated auxiliary inputs are impossible assuming the existence of pc-diO for circuits. We stress, however, that [20] only assumes the existence of a much weaker notion of *public-coin* SNARKs for their positive result. Therefore, the impossibility result of [6] is *not* applicable to [20].

implies indistinguishability obfuscation for Turing machines with unbounded input length.

Applications. We briefly highlight a few novel applications of our main result:

- *On-the-fly secure computation*: MI-FE for unbounded inputs naturally yields a new notion of *on-the-fly* secure multiparty computation in the correlated randomness model where new parties can join the system *dynamically* at any point in time. To the best of our knowledge, no prior solution for secure computation (even in the interactive setting) exhibits this property.
  In order to further explain this result, we first recall an application of MI-FE for bounded inputs to secure computation on the web [14] (this is implicit in [10]): consider a group of n parties who wish to jointly compute a function f over their private inputs using a web server. Given an MI-FE scheme that supports f, each party can simply send an encryption of its input x_i w.r.t. to its own encryption key to the server. Upon receiving all the ciphertexts, the server can then use a decryption key K_f (which is given to it as part of a correlated randomness setup) to compute $f(x_1, \ldots, x_n)$. Note that unlike the traditional solutions for secure computation that require simultaneous participation from each player, this solution is completely non-interactive and asynchronous (during the computation phase), which is particularly appealing for applications over the web.[2]
  Note that in the above application, since the number of inputs for the MI-FE scheme are a priori bounded, it means that the number of parties must also be bounded at the time of correlated randomness setup. In contrast, by plugging in our new MI-FE scheme for unbounded inputs in the above template, we now no longer need to fix the number of users in advance, and hence new users can join the system on "on-the-fly." In particular, the same decryption key K_f that was computed during the correlated randomness setup phase can still be used even when new users are dynamically added to the system.
- *Computing on encrypted databases of dynamic size*: In a similar vein, our MI-FE scheme enables arbitrary Turing machine computations on an encrypted database where the size of the database is not fixed a priori and can be increased dynamically.[3] Concretely, given a database of initial size n, we can start by encrypting each record separately. If the database owner wishes to later add new records to the database, then she can simply encrypt these records afresh and then add them to the existing encrypted database. Note that a decryption key K_M that was issued previously can still be used to compute on the updated database since we allow for Turing machines of unbounded input length.

[2] One should note, however, that due to its non-interactive nature, this solution only achieves a weaker indistinguishability-based notion of security for secure computation where the adversary also gets access to the residual function $f(\boldsymbol{x}_H^b, \cdot)$. Here $(\boldsymbol{x}_H^0, \boldsymbol{x}_H^1)$ are vectors of inputs of the honest parties.

[3] The same idea can be naturally extended to multiple databases.

We finally remark that this solution also facilitates "flexible" computations: suppose that a user is only interested in learning the output of M on a subset S of the records of size (say) $\ell \ll n$. Then, if we were to jointly compute on the entire encrypted database, the computation time would be proportional to n. In contrast, our scheme facilitates selective (joint) decryption of the encryptions of the records in S; as such, the running time of the resulting computation is only proportional to ℓ.

1.2 Technical Overview

In this work, we consider the indistinguishability-based selective security model for unbounded arity multi-input functional encryption[4]. The starting point for our construction is the MiFE scheme for bounded arity functions [10]. Similar to their work, in our construction, each ciphertext will consist of two ciphertexts under pk_1 and pk_2, and some other elements specific to the particular encryption key used. At a high level, a function key for a turing machine M will be an obfuscation of a machine which receives a collection of ciphertexts, decrypts them using sk_1, and returns the output of the turing machine on the decrypted messages. Before we decrypt the ciphertext with sk_1, we also need to have a some check that the given ciphertext is a valid encryption corresponding to a certain index. This check needs to be performed by the functional key for the turing machine M. Moreover, there is a distinct encryption key for each index and we do not have any a-priori bound on the number of inputs to our functions. Hence, the kinds of potential checks which need to be performed are unbounded in number. Dealing with unbounded unbounded number of encryption keys is the main technical challenge we face in designing an unbounded arity multi-input functional encryption scheme. We describe this in more detail below.

In the indistinguishability based security game of MiFE, the adversary can query for any polynomial number of encryption keys and is capable of encrypting under those. Finally, it provides the two challenge vectors. For the security proof to go through, we need to switch-off all encryption keys which are not asked by the adversary. The construction of [10] achieves this by having a separate "flag" value for each encryption key; this flag is part of the public parameters and also hardcoded in all the function keys that are given out. This approach obviously does not work in our case because we are dealing with unbounded number of encryption keys. This is one of the main technical difficulties which we face in extending the construction of MiFE for bounded arity to our case. We would like to point out that these problems can be solved easily using diO along with signatures, but we want our construction to only rely on pc-diO.

At a high level, we solve this issue of handling and blocking the above mentioned unbounded number of keys as follows: The public parameters of our scheme will consist of a pseudorandom string $u = G(z)$ and a random string

[4] It was shown in [10] that simulation-based security even for bounded arity MiFE implies the strong notion of black-box obfuscation. Hence, we do not consider that notion in this paper.

α. An encryption key EK_i for index i will consist of a proof that either there exists a z such that $u = G(z)$ or there exists a string x such that $x[j] = i$ and $\alpha = h(x)$, where h is a collision resistant hash function. Our programs only contain u and α hardcoded and hence their size is independent of the number of keys we can handle. In our sequence of hybrids, we will change u to be a random string and $\alpha = h(\mathrm{I})$, where I denotes the indices of the keys given out to the adversary. The encryption keys (which are asked by the adversary) will now use a proof for the second part of the statement and we show that a valid proof for an encryption key which is not given out to the adversary leads to a collision in the hash function.

Another issue which occurs is relating to the challenge ciphertexts for the indices for which the encryption key is not given to the adversary. Consider the setting when there is some index, say i^*, in challenge vector such that EK_{i^*} is secret. In the security game of MiFE we are guaranteed that output of M on any subset of either of the challenge ciphertexts along with any collection of the ciphertexts which the adversary can generate, is identical for both the challenge vectors. As mentioned before, for security proof to go through we need to ensure that for i^*, there should only exist the encryption of $x^0_{i^*}$ and $x^1_{i^*}$ (which are the challenge messages) and nothing else. Otherwise, if the adversary is able to come up with a ciphertext of $y^* \neq x^b_{i^*}$, he might be able to distinguish trivially. This is because we do not have any output restriction corresponding to y^*. In other words, we do not want to rule out all ciphertexts under EK_{i^*}; we want to rule out everything except $x^0_{i^*}$ and $x^1_{i^*}$. In the MiFE for bounded inputs [10], this problem was solved by hardcoding these specific challenge ciphertexts in public parameters as well as function keys. In our case, this will clearly not work since there is no bound on length of challenge vectors. We again use ideas involving collision resistant hash functions to deal with these issues. In particular, we hash the challenge vector and include a commitment to this hash value as part of the public parameters as well as the function keys. Note that we can do this because we only need to prove the selective security of our scheme.

We note that since collision resistant hash-functions have no trapdoor secret information, they work well with pc-diO assumption. We will crucially rely on pc-diO property while changing the program from using sk_1 to sk_2. Note that there would exist inputs on which the programs would differ, but these inputs would be hard to find for any PPT adversary even given all the randomness used to sample the two programs.

MiFE with unbounded arity implies iO for turing machines with unbounded inputs. First we recall the proof for the fact that MiFE with bounded number of inputs implies iO for circuits. To construct an iO for circuit C with n inputs, consider an MiFE scheme which supports arity $n + 1$. Under the first index EK_1, encrypt C and under keys $\{2, \ldots, n+1\}$ give out encryptions of both 0 and 1 under each index. Also, the secret key corresponding to universal circuit is given out. For our case, consider the setting of two encryption keys EK_1 and EK_2. We give out the encryption of the machine M under EK_1 and also the

key EK_2. That is, we are in the partial public key setting. We also give out the secret key corresponding to a universal turing machine which accepts inputs of unbounded length. Now, the user can encrypt inputs of unbounded length under the key EK_2 by encrypting his input bit by bit. Note that our construction allows encryption of multiple inputs under the same key.

2 Preliminaries

In this section, we describe the primitives used in our construction. Let λ be the security parameter.

2.1 Public-Coin Differing-Inputs Obfuscation

The notion of public coin differing-inputs obfuscation (pc-diO) was recently introduced by Yuval Ishai, Omkant Pandey, and Amit Sahai [15].

Let $\mathbb{N}$ denote the set of all natural numbers. We denote by $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$, a parameterized collection of Turing machines (TM) such that $\mathcal{M}_\lambda$ is the set of all TMs of size at most λ which halt within polynomial number of steps on all inputs. For $x \in \{0,1\}^*$, if M halts on input x, we denote by $\mathsf{steps}(\mathsf{M}, x)$ the number of steps M takes to output $\mathsf{M}(x)$. We also adopt the convention that the output $\mathsf{M}(x)$ includes the number of steps M takes on x, in addition to the actual output. The following definitions are taken almost verbatim from [15].

Definition 1 (Public-Coin Differing-Inputs Sampler for TMs). *An efficient non-uniform sampling algorithm* $\mathsf{Sam} = \{\mathsf{Sam}_\lambda\}$ *is called a public-coin differing-inputs sampler for the parameterized collection of TMs* $\mathcal{M} = \{\mathcal{M}_\lambda\}$ *if the output of* Sam_λ *is always a pair of Turing Machines* $(\mathsf{M}_0, \mathsf{M}_1) \in \mathcal{M}_\lambda \times \mathcal{M}_\lambda$ *such that* $|\mathsf{M}_0| = |\mathsf{M}_1|$ *and for all efficient non-uniform adversaries* $\mathcal{A} = \{\mathcal{A}_\lambda\}$*, there exists a negligible function* ϵ *such that for all* $\lambda \in \mathbb{N}$ *:*

$$\Pr_r\left[\begin{array}{c}\mathsf{M}_0(x) \neq \mathsf{M}_1(x) \wedge \\ \mathsf{steps}(\mathsf{M}_0, x) = \mathsf{steps}(\mathsf{M}_1, x) = t\end{array}\middle|\begin{array}{c}(\mathsf{M}_0, \mathsf{M}_1) \leftarrow \mathsf{Sam}_\lambda(r); \\ (x, 1^t) \leftarrow \mathcal{A}_\lambda(r)\end{array}\right] \leqslant \epsilon(\lambda)$$

By requiring $\mathcal{A}_\lambda$ *to output* 1^t*, we rule out all inputs* x *for which* $\mathsf{M}_0, \mathsf{M}_1$ *may take more than polynomial steps.*

Definition 2 (Public-Coin Differing-Inputs Obfuscator for TMs). *A uniform* PPT *algorithm* $\mathcal{O}$ *is called a public-coin differing-inputs obfuscator for the parameterized collection of TMs* $\mathcal{M} = \{\mathcal{M}_\lambda\}$ *if the following requirements hold:*

- **Correctness:** $\forall\lambda, \forall\mathsf{M} \in \mathcal{M}_\lambda, \forall x \in \{0,1\}^*$*, we have* $\Pr[\mathsf{M}'(x) = \mathsf{M}(x) : \mathsf{M}' \leftarrow \mathcal{O}(1^\lambda, \mathsf{M})] = 1$.
- **Security:** *For every public-coin differing-inputs sampler* $\mathsf{Sam} = \{\mathsf{Sam}_\lambda\}$ *for the collection* $\mathcal{M}$*, for every efficient non-uniform distinguishing algorithm* $\mathcal{D} = \{\mathcal{D}_\lambda\}$*, there exists a negligible function* ϵ *such that for all* λ *:*

$$\left|\begin{array}{l}\Pr[\mathcal{D}_\lambda(r, \mathsf{M}') = 1 : (\mathsf{M}_0, \mathsf{M}_1) \leftarrow \mathsf{Sam}_\lambda(r), \mathsf{M}' \leftarrow \mathcal{O}(1^\lambda, \mathsf{M}_0)] - \\ \Pr[\mathcal{D}_\lambda(r, \mathsf{M}') = 1 : (\mathsf{M}_0, \mathsf{M}_1) \leftarrow \mathsf{Sam}_\lambda(r), \mathsf{M}' \leftarrow \mathcal{O}(1^\lambda, \mathsf{M}_1)]\end{array}\right| \leq \epsilon(\lambda)$$

where the probability is taken over r and the coins of $\mathcal{O}$.

- **Succinctness and input-specific running time:** *There exists a (global) polynomial s' such that for all λ, for all $\mathsf{M} \in \mathcal{M}_\lambda$, for all $\mathsf{M}' \leftarrow \mathcal{O}(1^\lambda, \mathsf{M})$, and for all $x \in \{0,1\}^*$, $\mathsf{steps}(\mathsf{M}', x) \leqslant s'(\lambda, \mathsf{steps}(\mathsf{M}, x))$.*

We note that the size of the obfuscated machine M' is always bounded by the running time of $\mathcal{O}$ which is polynomial in λ. More importantly, the size of M' is independent of the running time of M. This holds even if we consider TMs which always run in polynomial time. This is because the polynomial bounding the running time of $\mathcal{O}$ is independent of the collection $\mathcal{M}$ being obfuscated. It is easy to obtain a uniform formulation from our current definitions.

2.2 Non Interactive Proof Systems

We start with the syntax and formal definition of a non-interactive proof system. Then, we give the definition of non-interactive witness indistinguishable proofs (NIWI) and strong non-interactive witness indistinguishable proofs (sNIWI).

Syntax : Let R be an efficiently computable relation that consists of pairs (x, w), where x is called the statement and w is the witness. Let L denote the language consisting of statements in R. A non-interactive proof system for a language L consists of the following algorithms:

- **Setup** $\mathsf{CRSGen}(1^\lambda)$ is a PPT algorithm that takes as input the security parameter λ and outputs a common reference string crs.
- **Prover** $\mathsf{Prove}(\mathsf{crs}, x, w)$ is a PPT algorithm that takes as input the common reference string crs, a statement x and a witness w. If $(x, w) \in R$, it produces a proof string π. Else, it outputs fail.
- **Verifier** $\mathsf{Verify}(\mathsf{crs}, x, \pi)$ is a PPT algorithm that takes as input the common reference string crs and a statement x with a corresponding proof π. It outputs 1 if the proof is valid, and 0 otherwise.

Definition 3 (Non-interactive Proof System). *A non-interactive proof system* $(\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Verify})$ *for a language L with a* PPT *relation R satisfies the following properties:*

- **Perfect Completeness :** *For every $(x, w) \in R$, it holds that*

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, x, \mathsf{Prove}(\mathsf{crs}, x, w))\right] = 1$$

  where $\mathsf{crs} \xleftarrow{\$} \mathsf{CRSGen}(1^\lambda)$, and the probability is taken over the coins of CRSGen, Prove *and* Verify.
- **Statistical Soundness:** *For every adversary $\mathcal{A}$, it holds that*

$$\Pr\left[x \notin L \wedge \mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 \;\middle|\; \mathsf{crs} \leftarrow \mathsf{CRSGen}(1^\lambda); (x, \pi) \leftarrow \mathcal{A}(\mathsf{crs})\right] \leqslant \mathsf{negl}(\lambda)$$

If the soundness property only holds against PPT adversaries, then we call it an argument system.

Definition 4. *(Strong Witness Indistinguishability* sNIWI*). Given a non-interactive proof system* $(\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Verify})$ *for a language* L *with a* PPT *relation* R*, let* $\mathcal{D}_0$ *and* $\mathcal{D}_1$ *be distributions which output an instance-witness pair* (x, w)*. We say that the proof system is strong witness-indistinguishable if for every adversary* $\mathcal{A}$ *and for all* PPT *distinguishers* D'*, it holds that*

$$\textit{If } \left| \Pr[D'(x) = 1 | (x, w) \leftarrow \mathcal{D}_0(1^\lambda)] - \Pr[D'(x) = 1 | (x, w) \leftarrow \mathcal{D}_1(1^\lambda)] \right| \leqslant \mathsf{negl}(\lambda)$$

$$\textit{Then } \begin{array}{l} |\Pr[\mathcal{A}(\mathsf{crs}, x, \mathsf{Prove}(\mathsf{crs}, x, w)) = 1 | (x, w) \leftarrow \mathcal{D}_0(1^\lambda)] - \\ \Pr[\mathcal{A}(\mathsf{crs}, x, \mathsf{Prove}(\mathsf{crs}, x, w)) = 1 | (x, w) \leftarrow \mathcal{D}_1(1^\lambda)]| \leqslant \mathsf{negl}(\lambda) \end{array}$$

The proof system of [7] is a strong non-interactive witness indistinguishable proof system.

2.3 Collision Resistent Hash Functions.

In this section, we describe the collision resistant hash functions mapping arbitrary polynomial length strings to $\{0,1\}^\lambda$. We begin by defining a family of collision resistant hash functions mapping 2λ length strings to λ length strings.

Definition 5. *Consider a family of hash functions* $\mathcal{H}'_\lambda$ *such that every* $h' \in \mathcal{H}'_\lambda$ *maps* $\{0,1\}^{2\lambda}$ *to* $\{0,1\}^\lambda$*.* $\mathcal{H}'_\lambda$ *is said to be a collision resistant hash family if for every* PPT *adversary* $\mathcal{A}$*,*

$$\Pr\left[h' \xleftarrow{\$} \mathcal{H}'_\lambda; (x, y) \leftarrow \mathcal{A}(h'); h'(x) = h'(y)\right] \leqslant \mathsf{negl}(\lambda)$$

In our scheme, we will need hash functions which hash unbounded length strings to $\{0,1\}^\lambda$. We describe these next, followed by a simple construction using Merkle trees [18]. In our construction, each block will consists of λ bits. Note that it is sufficient to consider a hash family hashing 2^λ blocks to λ bits, i.e., hashing strings of length at most $\lambda 2^\lambda$ to λ bits.

Definition 6. *[Family of collision resistant hash functions for unbounded length strings] Consider a family of hash functions* $\mathcal{H}_\lambda$ *such that every* $h \in \mathcal{H}_\lambda$ *maps strings of length* at most $\{0,1\}^{\lambda 2^\lambda}$ *to* $\{0,1\}^\lambda$*. Additionally, it supports the following functions:*

- $\mathsf{H.Open}(h, x, i, y)$*: Given a hash function key* h*, a string* $x \in \{0,1\}^*$ *such that* $|x| \leqslant \lambda 2^\lambda$*, an index* $i \in [|x|]$*, and* $y \in \{0,1\}^\lambda$*, it outputs a short proof* $\gamma \in \{0,1\}^{\lambda^2}$ *that* $x[i] = y$*.*
- $\mathsf{H.Verify}(h, y, u, \gamma, i)$*: Given a hash function key* h*, a string* $y \in \{0,1\}^\lambda$*, a string* $u \in \{0,1\}^\lambda$*, a string* $\gamma \in \{0,1\}^{\lambda^2}$ *and an index* $i \in [2^\lambda]$*, it outputs either* accept *or* reject*. This algorithm essentially verifies that there exists a* x *such that* $y = h(x)$ *and* $x[i] = u$*.*

For security it is required to satisfy the following property of collision resistance.
Collision Resistance. *The hash function family* $\mathcal{H}_\lambda$ *is said to be collision resistant if for every* PPT *adversary* $\mathcal{A}$*,*

$$\Pr\left[h \xleftarrow{\$} \mathcal{H}_\lambda; (x, u, \gamma, i) \leftarrow \mathcal{A}(h) \text{ s.t. } h(x) = y; x[i] \neq u; \mathsf{H.Verify}(h, y, u, \gamma, i) = \textit{accept}\right] \leqslant \mathsf{negl}(\lambda)$$

Construction : The above described scheme can be constructed by a merkle hash tree based construction on standard collision resistant hash functions of Definition 5.

3 Unbounded Arity Multi-Input Functional Encryption

Multi-input functional encryption(MiFE) for bounded arity functions (or circuits) was first introduced in [11,12]. In other words, for any bound n on the number of inputs, they designed an encryption scheme such that the owner of the master secret key MSK, can generate function keys sk_f corresponding to functions f accepting n inputs. That is, sk_f computes on $\mathsf{CT}_1, \ldots, \mathsf{CT}_\mathsf{n}$ to produce $f(x_1, \ldots, x_n)$ as output where CT_i is an encryption of x_i. In this work, we remove the a-priori bound n on the cardinality of the function.

In this work, we consider multi-input functional encryption for functions which accept unbounded number of inputs. That is, the input length is not bounded at the time of function key generation. Since we are dealing with FE for functions accepting unbounded number of inputs, in essence, we are dealing with TMs (with unbounded inputs) instead of circuits (with bounded inputs). Similar to MiFE with bounded inputs which allows for multi-party computation with bounded number of players, our scheme allows multiparty computation with a-priori unbounded number of parties. In other words, our scheme allows for more parties to join on-the-fly even after function keys have been given out. Moreover, similar to original MiFE, we want that each party is able to encrypt under different encryption keys, i.e., we want to support unbounded number of encryption keys. We want to achieve all this while keeping the size of the public parameters, master secret key as well as the function keys to be bounded by some fixed polynomial in the security parameter.

As mentioned before, we consider unbounded number of encryption keys, some of which may be made public, while rest are kept secret. When all the encryption keys corresponding to the challenge ciphertexts of the adversary are public, it represents the "public-key setting". On the other hand, when none of the keys are made public, it is called the "secret-key" setting. Our modeling allows us to capture the general setting when any polynomial number of keys can be made public. This can correspond to any subset of the keys associated with the challenge ciphertexts as well as any number of other keys. Note that we have (any) unbounded polynomial number of keys in our system unlike previous cases, where the only keys are the ones associated with challenge ciphertext.

As another level of generality, we allow that the turing machines or the functions can be invoked with ciphertexts corresponding to any subset of the encryption keys. Hence, if CT_j is an encryption of x_j under key EK_{i_j} then sk_M on $\mathsf{CT}_1, \ldots, \mathsf{CT}_\mathsf{n}$ computes $\mathsf{M}((x_1, i_1), \ldots, (x_n, i_n))$. Here sk_M corresponds to the key for the turing machine M.

Now, we first present the syntax and correctness requirements for unbounded arity multi-input functional encryption in Section 3.1 and then present the security definition in Section 3.2.

3.1 Syntax

Let $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$, $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathcal{K} = \{\mathcal{K}_\lambda\}_{\lambda\in\mathbb{N}}$ be ensembles where each $\mathcal{X}_\lambda, \mathcal{Y}_\lambda, \mathcal{K}_\lambda \subseteq [2^\lambda]$. Let $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ be an ensemble such that each $\mathsf{M} \in \mathcal{M}_\lambda$ is a turing machine accepting an (a-priori) unbounded polynomial (in λ) length of inputs. Each input string to a function $\mathsf{M} \in \mathcal{M}_\lambda$ is a tuple over $\mathcal{X}_\lambda \times \mathcal{K}_\lambda$. A turing machine $\mathsf{M} \in \mathcal{M}_\lambda$, on input a n length tuple $((x_1, i_1), (x_2, i_2), \ldots, (x_n, i_n))$ outputs $M((x_1, i_1), (x_2, i_2), \ldots, (x_n, i_n)) \in \mathcal{Y}_\lambda$, where $(x_j, i_j) \in \mathcal{X}_\lambda \times \mathcal{K}_\lambda$ for all $j \in [n]$ and $n(\lambda)$ is any arbitrary polynomial in λ.

An unbounded arity multi-input functional encryption scheme FE for $\mathcal{M}$ consists of five algorithms ($\mathsf{FE.Setup}$, $\mathsf{FE.EncKeyGen}$, $\mathsf{FE.Enc}$, $\mathsf{FE.FuncKeyGen}$, $\mathsf{FE.Dec}$) described below.

- **Setup** $\mathsf{FE.Setup}(1^\lambda)$ is a PPT algorithm that takes as input the security parameter λ and outputs the public parameters PP and the master secret key MSK.
- **Encryption Key Generation** $\mathsf{FE.EncKeyGen}(\mathsf{PP}, i, \mathsf{MSK})$ is a PPT algorithm that takes as input the public parameters PP, an index $i \in \mathcal{K}_\lambda$ and master secret key MSK, and outputs the encryption key EK_i corresponding to index i.
- **Encryption** $\mathsf{FE.Enc}(\mathsf{PP}, \mathsf{EK}_i, x)$ is a PPT algorithm that takes as input public parameters PP, an encryption key EK_i and an input message $x \in \mathcal{X}_\lambda$ and outputs a ciphertext CT encrypting (x, i). Note that the ciphertext also incorporates the index of the encryption key.
- **Function Key Generation** $\mathsf{FE.FuncKeyGen}(\mathsf{PP}, \mathsf{MSK}, \mathsf{M})$ is a PPT algorithm that takes as input public parameters PP, the master secret key MSK, a turing machine $\mathsf{M} \in \mathcal{M}_\lambda$ and outputs a corresponding secret key SK_M.
- **Decryption** $\mathsf{FE.Dec}(\mathsf{SK}_\mathsf{M}, \mathsf{CT}_1, \mathsf{CT}_2, \ldots, \mathsf{CT}_n)$ is a deterministic algorithm that takes as input a secret key SK_M and a set of ciphertexts $\mathsf{CT}_1, \ldots, \mathsf{CT}_n$ as input and outputs a string $y \in \mathcal{Y}_\lambda$. Note that there is no a-priori bound on n.

Definition 7 (Correctness). *An unbounded arity multi-input functional encryption scheme* FE *for* $\mathcal{M}$ *is correct if* $\forall \mathsf{M} \in \mathcal{M}_\lambda$, $\forall n$ *s.t* $n = p(\lambda)$, *for some polynomial* p, *all* $(x_1, x_2, \ldots, x_n) \in \mathcal{X}_\lambda^n$ *and all* $\mathrm{I} = (i_1, \ldots, i_n) \in \mathcal{K}_\lambda^n$ *:*

$$\Pr\left[\begin{array}{l}(\mathsf{PP}, \mathsf{MSK}) \leftarrow \mathsf{FE.Setup}(1^\lambda); \mathsf{EK}_\mathrm{I} \leftarrow \mathsf{FE.EncKeyGen}(\mathsf{PP}, \mathrm{I}, \mathsf{MSK});\\ \mathsf{SK}_\mathsf{M} \leftarrow \mathsf{FE.FuncKeyGen}(\mathsf{PP}, \mathsf{MSK}, \mathsf{M});\\ \mathsf{FE.Dec}(\mathsf{SK}_\mathsf{M}, \mathsf{FE.Enc}(\mathsf{PP}, \mathsf{EK}_{i_1}, x_1), \ldots, \mathsf{FE.Enc}(\mathsf{PP}, \mathsf{EK}_{i_n}, x_n)) \neq\\ \mathsf{M}((x_1, i_1), \ldots, (x_n, i_n))\end{array}\right] \leqslant \mathsf{negl}(\lambda)$$

Here, EK_I denotes a set of encryption keys corresponding to the indices in the set I. For each $i \in \mathrm{I}$, we run $\mathsf{FE.EncKeyGen}(\mathsf{PP}, i, \mathsf{MSK})$ and we denote that in short by $\mathsf{FE.EncKeyGen}(\mathsf{PP}, \mathrm{I}, \mathsf{MSK})$.

3.2 Security Definition

We consider indistinguishability based selective security (or IND-security, in short) for unbounded arity multi-input functional encryption. This notion will be defined very similar to the security definition in original MiFE papers [11,12]. We begin by recalling this notion.

Let us consider the simple case of 2-ary functions $f(\cdot,\cdot)$ such that adversary requests the function key for f as well as the encryption key for the second index. Let the challenge ciphertext be (x^0, y^0) and (x^1, y^1). For the indistinguishability of challenge vectors, first condition required is that $f(x^0, y^0) = f(x^1, y^1)$. Moreover, since the adversary has the encryption key for the second index, he can encrypt any message corresponding to the second index. Hence, if there exists a y^* such that $f(x^0, y^*) \neq f(x^1, y^*)$, then distinguishing is easy! Hence, they additionally require that $f(x^0, \cdot) = f(x^1, \cdot)$ for all the function queries made by the adversary. That is, the function queries made have to be compatible with the encryption keys requested by the adversary; otherwise the task of distinguishing is trivial.

Similar to this notion, since in our case as well, the adversary can request any subset of the encryption keys, we require that the function key queries are compatible with encryption key queries. Since we allow the turing machine to be invoked with any subset of the key indices and potentially unbounded number of key indices, this condition is much more involved in our setting. At a high level, we require that the function outputs should be identical for any subset of the two challenge inputs combined with any vector of inputs for indices for which adversary has the encryption keys. More formally, we define the notion of I-compatibility as follows:

Definition 8 (I-Compatibility). *Let* $\{\mathsf{M}\}$ *be any set of turing machines such that every turing machine* M *in the set belongs to* $\mathcal{M}_\lambda$*. Let* $\mathrm{I} \subseteq \mathcal{K}_\lambda$ *such that* $|\mathrm{I}| = q(\lambda)$ *for some polynomial* q*. Let* $\boldsymbol{X}^0$ *and* $\boldsymbol{X}^1$ *be a pair of input vectors, where* $\boldsymbol{X}^b = \{(x_1^b, k_1), (x_2^b, k_2), \ldots, (x_n^b, k_n)\}$ *such that* $n = p(\lambda)$ *for some polynomial* p*. We say that* $\{\mathsf{M}\}$ *and* $(\boldsymbol{X}^0, \boldsymbol{X}^1)$ *are* I*-compatible if they satisfy the following property:*

- *For every* $\mathsf{M} \in \{\mathsf{M}\}$*, every* $\mathrm{I}' = \{i_1, \ldots, i_\alpha\} \subseteq \mathrm{I}$*, every* $\mathrm{J} = \{j_1, \ldots, j_\beta\} \subseteq [n]$*, and every* $y_1, \ldots, y_\alpha \in \mathcal{X}_\lambda$ *and every permutation* $\pi : [\alpha+\beta] \to [\alpha+\beta]$ *:*

$$\mathsf{M}\Big(\pi\Big((y_1, i_1), (y_2, i_2), \ldots, (y_\alpha, i_\alpha), (x^0_{j_1}, k_{j_1}), (x^0_{j_2}, k_{j_2}), \ldots, (x^0_{j_\beta}, k_{j_\beta})\Big)\Big) =$$
$$\mathsf{M}\Big(\pi\Big((y_1, i_1), (y_2, i_2), \ldots, (y_\alpha, i_\alpha), (x^1_{j_1}, k_{j_1}), (x^1_{j_2}, k_{j_2}), \ldots, (x^1_{j_\beta}, k_{j_\beta})\Big)\Big)$$

  Here, $\pi(a_1, a_2, \ldots, a_{\alpha+\beta})$ *denotes the permutation of the elements* $a_1, \ldots, a_{\alpha+\beta}$*.*

We now present our formal security definition for IND-secure unbounded arity multi-input functional encryption.

Selective IND-Secure MiFE . This is defined using the following game between the challenger and the adversary.

Definition 9 (Indistinguishability-Based Selective Security). *We say that an unbounded arity multi-input functional encryption scheme* FE *for* $\mathcal{M}$ *is* IND*-secure if for every* PPT *adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$*, for all polynomials* p, q *and for all* $m = p(\lambda)$ *and for all* $n = q(\lambda)$*, the advantage of* $\mathcal{A}$ *defined as*

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

is $\mathsf{negl}(\lambda)$ *where the experiment is defined below.*

Experiment $\mathsf{IND}_{\mathcal{A}}^{\mathsf{FE}}(1^\lambda)$:

$(\mathrm{I}, \boldsymbol{X}^0, \boldsymbol{X}^1, st_0) \leftarrow \mathcal{A}_0(1^\lambda)$ where $|\mathrm{I}| = m$; $\boldsymbol{X}^\ell = \{(x_1^\ell, k_1), (x_2^\ell, k_2), \ldots, (x_n^\ell, k_n)\}$
$(\mathsf{PP}, \mathsf{MSK}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$
Compute $\mathsf{EK}_i \leftarrow \mathsf{FE.EncKeyGen}(\mathsf{PP}, i, \mathsf{MSK})$, $\forall i \in \mathrm{I}$. Let $\mathsf{EK}_\mathrm{I} = \{\mathsf{EK}_i\}_{i \in \mathrm{I}}$.
$b \xleftarrow{\$} \{0,1\}$; $\mathsf{CT}_i \leftarrow \mathsf{FE.Enc}(\mathsf{EK}_{k_i}, x_i^b)$, $\forall i \in [n]$. Let $\mathbf{CT} = \{\mathsf{CT}_1, \ldots, \mathsf{CT}_n\}$
$b' \leftarrow \mathcal{A}_1^{\mathsf{FE.FuncKeyGen}(\mathsf{PP},\mathsf{MSK},\cdot)}(st_0, \mathsf{PP}, \mathsf{EK}_\mathrm{I}, \mathbf{CT})$
Output: $(b = b')$

Fig. 1

In the above experiment, we require :

- *Let* $\{\mathsf{M}\}$ *denote the entire set of function 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* $\{\mathsf{M}\}$*.*

4 A Construction from Public-Coin Differing-Inputs Obfuscation

Notation : Without loss of generality, let's assume that every plaintext message and encryption key index is of length λ where λ denotes the security parameter of our scheme. Let $(\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Verify})$ be a statistically sound, non-interactive strong witness-indistinguishable proof system for NP, $\mathcal{O}$ denote a public coin differing-inputs obfuscator, $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ be a semantically secure public key encryption scheme, com be a statistically binding and computationally hiding commitment scheme and G be a pseudorandom generator from $\{0,1\}^\lambda$ to $\{0,1\}^{2\lambda}$. Without loss of generality, let's say com commits to a string bit-by-bit and uses randomness of length λ to commit to a single bit. Let $\{H_\lambda\}$ be a family of merkle hash functions such that every $h \in H_\lambda$ maps strings from $\{0,1\}^{\lambda 2^\lambda}$ to $\{0,1\}^\lambda$. That is, the merkle tree has depth λ.
We now describe our scheme $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.EncKeyGen}, \mathsf{FE.Enc}, \mathsf{FE.FuncKeyGen}, \mathsf{FE.Dec})$ as follows:

– **Setup** $\mathsf{FE.Setup}(1^\lambda)$ **:**
The setup algorithm first computes $\mathsf{crs} \leftarrow \mathsf{CRSGen}(1^\lambda)$. Next, it computes $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$, $(\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$, $(\mathsf{pk}_3, \mathsf{sk}_3) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$ and $(\mathsf{pk}_4, \mathsf{sk}_4) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$. Let $\alpha = \mathsf{com}(0^\lambda; u)$, $\beta_1 = \mathsf{com}(0^\lambda; u_1)$ and $\beta_2 = \mathsf{com}(0^\lambda; u_2)$ where u, u_1 and u_2 are random strings of length λ^2. Choose a hash function $h \leftarrow H_\lambda$. Choose $z \xleftarrow{\$} \{0,1\}^\lambda$ and compute $Z = G(z)$.

The public parameters are $\mathsf{PP} = (\mathsf{crs}, \mathsf{pk}_1, \mathsf{pk}_2, \mathsf{pk}_3, \mathsf{pk}_4, h, \alpha, \beta_1, \beta_2, Z)$.
The master secret key is $\mathsf{MSK} = (\mathsf{sk}_1, z, u, u_1, u_2)$.

– **Encryption Key Generation** $\mathsf{FE.EncKeyGen}(\mathsf{PP}, i, \mathsf{MSK})$ **:**
Given an index i, this algorithm first defines $b_i = z||0^\lambda||0^{\lambda^2}||0^{\lambda^2}||0^\lambda$. Then, it computes $d_i = \mathsf{PKE.Enc}(\mathsf{pk}_4, b_i; r)$ for some randomness r and $\sigma_i \leftarrow \mathsf{Prove}(\mathsf{crs}, st_i, w_i)$ for the statement that $st_i \in L_1$ using witness $w_i = (b_i, r)$ where $st_i = (d_i, i, \mathsf{pk}_4, \alpha, Z)$.
L_1 is defined corresponding to the relation R_1 defined below.

Relation R_1 :
Instance : $st_i = (d_i, i, \mathsf{pk}_4, \alpha, Z)$
Witness : $w = (b_i, r)$, where $b_i = z||\mathsf{hv}||\gamma||u||t$
$R_1(st_i, w) = 1$ if and only if the following conditions hold :
1. $d_i = \mathsf{PKE.Enc}(\mathsf{pk}_4, b_i; r)$ AND
2. The OR of the following statements must be true :
   (a) $G(z) = Z$
   (b) $\mathsf{H.Verify}(h, \mathsf{hv}, i, \gamma, t) = 1$ and $\mathsf{com}(\mathsf{hv}; u) = \alpha$

The output of the algorithm is the i^{th} encryption key $\mathsf{EK}_i = (\sigma_i, d_i, i)$, where σ_i is computed using witness for statements 1 and 2(a) of R_1.

– **Encryption** $\mathsf{FE.Enc}(\mathsf{PP}, \mathsf{EK}_i, x)$ **:**
To encrypt a message x with the i^{th} encryption key EK_i, the encryption algorithm first computes $c_1 = \mathsf{PKE.Enc}(\mathsf{pk}_1, x||i; r_1)$ and $c_2 = \mathsf{PKE.Enc}(\mathsf{pk}_2, x||i; r_2)$. Define string $a = x||i||r_1||0^{\lambda^2}||0^\lambda||0^{\lambda^2}||x||i||r_2||0^{\lambda^2}||0^\lambda||0^{\lambda^2}||0^\lambda$ and compute $c_3 = \mathsf{PKE.Enc}(\mathsf{pk}_3, a; r_3)$. Next, it computes a proof $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, y, w)$ for the statement that $y \in L_2$ using witness w where :
$y = (c_1, c_2, c_3, \mathsf{pk}_1, \mathsf{pk}_2, \mathsf{pk}_3, \mathsf{pk}_4, \beta_1, \beta_2, i, d_i, \alpha, Z)$
$w = (a, r_3, \sigma_i)$
L_2 is defined corresponding to the relation R_2 defined below.

Relation R_2 :
Instance : $y = (c_1, c_2, c_3, \mathsf{pk}_1, \mathsf{pk}_2, \mathsf{pk}_3, \mathsf{pk}_4, \beta_1, \beta_2, i, d_i, \alpha, Z)$
Witness : $w = (a, r_3, \sigma_i)$ where $a = x_1||i_1||r_1||u_1||\mathsf{hv}_1||\gamma_1||x_2||i_2||r_2||u_2||\mathsf{hv}_2||\gamma_2||t$
$R_2(y, w) = 1$ if and only if the following conditions hold :
1. $c_3 = \mathsf{PKE.Enc}(\mathsf{pk}_3, a; r_3)$ AND
2. The OR of the following two statements 2(a) and 2(b) is true :
   (a) The OR of the following two statements is true :

i. $(c_1 = \mathsf{PKE.Enc}(\mathsf{pk}_1, (x_1||i_1); r_1)$ AND $c_2 = \mathsf{PKE.Enc}(\mathsf{pk}_2, (x_1||i_1); r_2)$ AND $i_1 = i$ AND $\mathsf{Verify}(\mathsf{crs}, st_i, \sigma_i) = 1$ such that $st_i = (d_i, i, \mathsf{pk}_4, \alpha, Z) \in L_1)$; OR

ii. $(c_1 = \mathsf{PKE.Enc}(\mathsf{pk}_1, (x_2||i_2); r_1)$ AND $c_2 = \mathsf{PKE.Enc}(\mathsf{pk}_2, (x_2||i_2); r_2)$ AND $i_2 = i$ AND $\mathsf{Verify}(\mathsf{crs}, st_i, \sigma_i) = 1$ such that $st_i = (d_i, i, \mathsf{pk}_4, \alpha, Z) \in L_1)$;

(b) c_1,c_2 encrypt $(x_1||i_1)$,$(x_2||i_2)$ respectively, which may be different but then both β_1 and β_2 contain a hash of one of them (which may be different). That is,

i. $c_1 = \mathsf{PKE.Enc}(\mathsf{pk}_1, (x_1||i_1); r_1)$ AND $c_2 = \mathsf{PKE.Enc}(\mathsf{pk}_2, (x_2||i_2); r_2)$

ii. $\mathsf{H.Verify}(h, \mathsf{hv}_1, (x_1||i_1), \gamma_1, t) = 1$ AND $\beta_1 = \mathsf{com}(\mathsf{hv}_1; u_1)$ OR $\mathsf{H.Verify}(h, \mathsf{hv}_1, (x_2||i_2), \gamma_1, t) = 1$ AND $\beta_1 = \mathsf{com}(\mathsf{hv}_1; u_1)$

iii. $\mathsf{H.Verify}(h, \mathsf{hv}_2, (x_1||i_1), \gamma_2, t) = 1$ AND $\beta_2 = \mathsf{com}(\mathsf{hv}_2; u_2)$ OR $\mathsf{H.Verify}(h, \mathsf{hv}_2, (x_2||i_2), \gamma_2, t) = 1$ AND $\beta_2 = \mathsf{com}(\mathsf{hv}_2; u_2)$

The output of the algorithm is the ciphertext $\mathsf{CT} = (c_1, c_2, c_3, d_i, \pi, i)$. π is computed for the AND of statements 1 and 2(a)i of R_2.

– **Function Key Generation** $\mathsf{FE.FuncKeyGen}(\mathsf{PP}, \mathsf{MSK}, \mathsf{M})$ **:** The algorithm computes $\mathsf{SK}_\mathsf{M} = \mathcal{O}(\mathsf{G}_\mathsf{M})$ where the program G_M is defined as follows :

Program G_M

Input : $\mathsf{CT}_1, \mathsf{CT}_n, \ldots, \mathsf{CT}_n$
Constants : $(\mathsf{sk}_1, \mathsf{PP})$, i.e. $(\mathsf{sk}_1, (\mathsf{crs}, \mathsf{pk}_1, \mathsf{pk}_2, \mathsf{pk}_3, \mathsf{pk}_4, h, \alpha, \beta_1, \beta_2, Z))$

1. For every $i \in [n]$:
   (a) Parse $\mathsf{CT}_i = (c_{i,1}, c_{i,2}, c_{i,3}, d_{k_i}, \pi_i, k_i)$
   (b) Let $y_i = (c_{i,1}, c_{i,2}, c_{i,3}, \mathsf{pk}_1, \mathsf{pk}_2, \mathsf{pk}_3, \mathsf{pk}_4, \beta_1, \beta_2, k_i, d_{k_i}, \alpha, Z)$ be the statement corresponding to the proof string π_i. If $\mathsf{Verify}(\mathsf{crs}, y_i, \pi_i) = 0$, then stop and output $\perp$. Else, continue to the next step.
   (c) Compute $(x_i||k_i) = \mathsf{PKE.Dec}(\mathsf{sk}_1, c_{i,1})$
2. Output $\mathsf{M}((x_1, k_1), (x_2, k_2), \ldots, (x_n, k_n))$

Fig. 2

– **Decryption** $\mathsf{FE.Dec}(\mathsf{SK}_\mathsf{M}, \mathsf{CT}_1, \ldots, \mathsf{CT}_n)$ **:** It computes and outputs $\mathsf{SK}_\mathsf{M}(\mathsf{CT}_1, \ldots, \mathsf{CT}_n)$.

5 Security Proof

We now prove that the proposed scheme FE is selective IND-secure.

Theorem 2. *Let $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ be a parameterized collection of Turing machines (TM) such that $\mathcal{M}_\lambda$ is the set of all TMs of size at most λ which halt within polynomial number of steps on all inputs. Then, assuming there exists a*

public-coin differing-inputs obfuscator for the class $\mathcal{M}$, a non-interactive strong witness indistinguishable proof system, a public key encryption scheme, a non-interactive perfectly binding computationally hiding commitment scheme, a pseudorandom generator and a family of merkle hash functions, the proposed scheme FE *is a selective* IND*-secure MIFE scheme with unbounded arity for Turing machines in the class $\mathcal{M}$ according to definition 9.*

We will prove the above theorem via a series of hybrid experiments $\mathsf{H}_0, \ldots, \mathsf{H}_{20}$ where H_0 corresponds to the real world experiment with challenge bit $b = 0$ and H_{20} corresponds to the real world experiment with challenge bit $b = 1$.

- **Hybrid** H_0: This is the real experiment with challenge bit $b = 0$. The public parameters are
  $\mathsf{PP} = (\mathsf{crs}, \mathsf{pk}_1, \mathsf{pk}_2, \mathsf{pk}_3, \mathsf{pk}_4, h, \alpha, \beta_1, \beta_2, Z)$ such that $\alpha = \mathsf{com}(0^\lambda; u)$, $\beta_1 = \mathsf{com}(0^\lambda; u_1)$,$\beta_2 = \mathsf{com}(0^\lambda; u_2)$ and $Z = G(z)$, where $z \xleftarrow{\$} \{0,1\}^\lambda$.

- **Hybrid** H_1: This hybrid is identical to the previous hybrid except that β_1 and β_2 are computed differently. β_1 is computed as a commitment to hash of the string $s_1 = (x_1^0||k_1, \ldots, x_n^0||k_n)$ where $\{(x_1^0, k_1), \ldots, (x_n^0, k_n)\}$ is the challenge message vector $\boldsymbol{X}^0$. Similarly, β_2 is computed as a commitment to hash of the string $s_2 = (x_1^1||k_1, \ldots, x_n^1||k_n)$ where $\{(x_1^1, k_1), \ldots, (x_n^1, k_n)\}$ is the challenge message vector $\boldsymbol{X}^1$. That is, $\beta_1 = \mathsf{com}(h(s_1); u_1)$ and $\beta_2 = \mathsf{com}(h(s_2); u_2)$. There is no change in the way the challenge ciphertexts are computed.
  Note that s_1 and s_2 are padded with sufficient zeros to satisfy the input length constraint of the hash function.

- **Hybrid** H_2: This hybrid is identical to the previous hybrid except that we change the third component (c_3) in every challenge ciphertext. Let the i^{th} challenge ciphertext be $\mathsf{CT}_i = (c_{i,1}, c_{i,2}, c_{i,3}, d_{k_i}, \pi_i, k_i)$ for all $i \in [n]$. Let $s_1 = (x_1^0||k_1, \ldots, x_n^0||k_n)$ and $s_2 = (x_1^1||k_1, \ldots, x_n^1||k_n)$. In the previous hybrid $c_{i,3}$ is an encryption of $a_i = x_i^0||k_i||r_1||0^{\lambda^2}||0^\lambda||0^{\lambda^2}||x_i^0||k_i||r_2||0^{\lambda^2}||0^\lambda||0^{\lambda^2}||0^\lambda$. Now, a_i is changed to $a_i = x_i^0||k_i||r_1||u_1||h(s_1)||\gamma_{1,i}||x_i^1||k_i||r_2||u_2||h(s_2)||\gamma_{2,i}||i$ where $\gamma_{1,i}, \gamma_{2,i}$ are the openings for $h(s_1)$ and $h(s_2)$ w.r.t. $x_i^0||k_i$ and $x_i^1||k_i$, respectively. That is, $\gamma_{1,i} = \mathsf{H.Open}(h, s_1, i, x_i^0||k_i)$ and $\gamma_{2,i} = \mathsf{H.Open}(h, s_2, i, x_i^1||k_i)$. Since a_i has changed, consequently, ciphertext $c_{i,3}$ which is an encryption of a_i, witness w_i for π_i and proof π_i change as well for all $i \in [n]$. Note that for all challenge ciphertexts, π still uses the witness for statement 1 and 2(a).

- **Hybrid** H_3: This hybrid is identical to the previous hybrid except that we change the second component in every challenge ciphertext. Let the i^{th} challenge ciphertext be CT_i where $i \in [n]$. Let's parse $\mathsf{CT}_i = (c_{i,1}, c_{i,2}, c_{i,3}, d_{k_i}, \pi_i, k_i)$. We change $c_{i,2}$ to be an encryption of $x_i^1||k_i$. Further, π_i is now computed using the AND of statements 1 and 2(b) in the relation R_2.

– **Hybrid** H_4: This hybrid is identical to the previous hybrid except that α is computed as a commitment to hash of the string $s = (k_1, k_2, \ldots, k_m)$ where $\{k_1, \ldots, k_m\}$ is the set of indices I for which the adversary requests encryption keys. i.e $\alpha = \mathsf{com}(h(s); u)$.
Note that in this hybrid, for any encryption key EK_i, the proof σ_i is unchanged and is generated using the AND of statements 1 and 2(a).

– **Hybrid** H_5: This hybrid is identical to the previous hybrid except that we change the second component d_{k_i} for every encryption key EK_{k_i} that is given out to the adversary. First, let's denote $s = (k_1, \ldots, k_m)$ as in the previous hybrid. d_{k_i} is an encryption of $b_{k_i} = z||0^\lambda||0^{\lambda^2}||0^{\lambda^2}||0^\lambda$. Now, b_{k_i} is changed to $b_{k_i} = z||h(s)||\gamma_i||u_1||i$ where u_1 is the randomness used in the commitment of α and γ_i is the opening of the hash values in the merkle tree. That is, $\gamma_i = \mathsf{H.Open}(h, s, i, k_i)$. Consequently, d_{k_i} which is an encryption of b_{k_i} also changes. Since b_{k_i} has changed, the witness used in computing the proof σ_{k_i} has also changed. Note that σ_{k_i} still uses the witness for statements 1 and 2(a).

– **Hybrid** H_6: This hybrid is identical to the previous hybrid except that for every encryption key EK_{k_i} that is given out to the adversary, σ_{k_i} is now computed using the AND of statements 1 and 2(b) in the relation R_1.

– **Hybrid** H_7: This hybrid is identical to the previous hybrid except that in the public parameters Z is chosen to be a uniformly random string. Therefore, now $\mathsf{G}(z) \neq Z$ except with negligible probability.

– **Hybrid** H_8: Same as the previous hybrid except that the challenger sets the master secret key to have sk_2 instead of sk_1 and for every function key query M, the corresponding secret key SK_M is computed as $\mathsf{SK}_\mathsf{M} \leftarrow \mathcal{O}(\mathsf{G}'_\mathsf{M})$ where the program G'_M is the same as G_M except that :
  1. It has secret key sk_2 as a constant hardwired into it instead of sk_1.
  2. It decrypts the *second* component of each input ciphertext using sk_2. That is, in step 1(C), $x_i||k_i$ is computed as $x_i||k_i = \mathsf{PKE.Dec}(\mathsf{sk}_2, c_{i,2})$

– **Hybrid** H_9: This hybrid is identical to the previous hybrid except that in the public parameters Z is chosen to be the output of the pseudorandom generator applied on the seed z. That is, $Z = \mathsf{G}(z)$.

– **Hybrid** H_{10}: This hybrid is identical to the previous hybrid except that for every encryption key EK_{k_i} that is given out to the adversary, we change σ_{k_i} to now be computed using the AND of statements 1 and 2(a) in the relation R_1.

Remark: Note that statement 2(b) is true as well for all EK_{k_i} but we choose to use 2(a) due to the following technical difficulty. Observe that at this point we need to somehow change each $c_{i,1}$ to be an encryption of $x_i^1||k_i$ instead of $x_i^0||k_i$. When we make this switch, the statement 2(b) in R_2 is no longer

true. This is because β_1 will not be valid w.r.t. $c_{i,1}$ and $c_{i,2}$ since both are now encryptions of $x_i^1||k_i$. So we need to make statement 2(a) true for all challenge ciphertexts including the ones under some EK_{k_j} such that $k_j \notin \mathrm{I}$.

- **Hybrid** H_{11}: This hybrid is identical to the previous hybrid except that we change the first component in every challenge ciphertext. Let the i^{th} challenge ciphertext be CT_i where $i \in [n]$. Let's parse $\mathsf{CT}_i = (c_{i,1}, c_{i,2}, c_{i,3}, d_{k_i}, \pi_i, k_i)$. We change $c_{i,1}$ to be an encryption of $x_i^1||k_i$. Then, we change the proof π_i to be computed using the AND of statements 1 and 2(a) in the relation R_2.

- **Hybrid** H_{12}: This hybrid is identical to the previous hybrid except that β_1 is computed differently. β_1 is computed as a commitment to hash of the string $s_2 = (x_1^1||k_1, \ldots, x_n^1||k_n)$ where $\{(x_1^1, k_1), \ldots, (x_n^1, k_n)\}$ is the challenge message vector $\boldsymbol{X}^1$. That is, $\beta_1 = \mathsf{com}(h(s_2); u_1)$
  Note that s_2 is padded with sufficient zeros to satisfy the input length constraint of the hash function. There is no change in the way the challenge ciphertexts are computed.

- **Hybrid** H_{13}: This hybrid is identical to the previous hybrid except that we change the proof in every challenge ciphertext. Let the i^{th} challenge ciphertext be CT_i where $i \in [n]$. Let's parse $\mathsf{CT}_i = (c_{i,1}, c_{i,2}, c_{i,3}, d_{k_i}, \pi_i, k_i)$. We change π_i to now be computed using the AND of statements 1 and 2(b) in the relation R_2.

- **Hybrid** H_{14}: This hybrid is identical to the previous hybrid except that for every encryption key EK_{k_i} that is given out to the adversary, we change σ_{k_i} to now be computed using the AND of statements 1 and 2(b) in the relation R_1.

- **Hybrid** H_{15}: This hybrid is identical to the previous hybrid except that in the public parameters Z is chosen to be a uniformly random string.

- **Hybrid** H_{16}: This hybrid is identical to the previous hybrid except that the master secret key is set back to having sk_1 instead of sk_2 and for every function key query M, the corresponding secret key SK_{M} is computed using obfuscation of the original program G_{M}, i.e $\mathsf{SK}_{\mathsf{M}} \leftarrow \mathcal{O}(\mathsf{G}_{\mathsf{M}})$.

- **Hybrid** H_{17}: This hybrid is identical to the previous hybrid except we change Z to be the output of the pseudorandom generator applied on the seed z. That is, $Z = G(z)$.

- **Hybrid** H_{18}: This hybrid is identical to the previous hybrid except that for every encryption key EK_{k_i} that is given out to the adversary, σ_{k_i} is now computed using the AND of statements 1 and 2(a) in the relation R_1.

- **Hybrid** H_{19}: This hybrid is identical to the previous hybrid except that we change the second component d_{k_i} for every encryption key EK_{k_i} that is given

out to the adversary. We change b_{k_i} to be $b_{k_i} = z||0^\lambda||0^{\lambda^2}||0^{\lambda^2}||0^\lambda$ and consequently d_{k_i} also changes as it is the encryption of b_{k_i}. Since b_{k_i} has changed, the witness used in computing the proof σ_{k_i} has also changed. Note that σ_{k_i} still uses the witness for statements 1 and 2(a).

- **Hybrid** H_{20}: This hybrid is identical to the previous hybrid except that we change α to be a commitment to 0^λ. That is, $\alpha = \mathsf{com}(0^\lambda; u)$.

- **Hybrid** H_{21}: This hybrid is identical to the previous hybrid except that for every challenge ciphertext key CT_i that is given out to the adversary, π_i is now computed using the AND of statements 1 and 2(a) in the relation R_2.

- **Hybrid** H_{22}: This hybrid is identical to the previous hybrid except that we change the third component in every challenge ciphertext. Let the i^{th} challenge ciphertext be CT_i where $i \in [n]$. Let's parse $\mathsf{CT}_i = (c_{i,1}, c_{i,2}, c_{i,3}, d_{k_i}, \pi_i, k_i)$ where $c_{i,3}$ is an encryption of a_i. Now, a_i is changed to $a_i = x_i^1||k_i||r_1||0^{\lambda^2}||0^\lambda||0^{\lambda^2}||x_i^1||k_i||r_2||0^{\lambda^2}||0^\lambda||0^{\lambda^2}||0^\lambda$. Consequently, ciphertext $c_{i,3}$ which is an encryption of a_i will also change. Note that for all challenge ciphertexts, π still uses the witness for statement 1 and 2(a).

- **Hybrid** H_{23}: This hybrid is identical to the previous hybrid except that β_1 and β_2 are both computed to be commitments of 0^λ. That is, $\beta_1 = \mathsf{com}(0^\lambda; u_1)$ and $\beta_2 = \mathsf{com}(0^\lambda; u_2)$. This is identical to the real experiment with challenge bit $b = 1$.

Below we will prove that ($\mathsf{H}_0 \approx_c \mathsf{H}_1$), ($\mathsf{H}_1 \approx_c \mathsf{H}_2$), and ($\mathsf{H}_7 \approx_c \mathsf{H}_8$). The indistinguishability of other hybrids will follow along the same lines.

Lemma 1. *(*$\mathsf{H}_0 \approx_c \mathsf{H}_1$*). Assuming that* com *is a (computationally) hiding commitment scheme, the outputs of experiments* H_0 *and* H_1 *are computationally indistinguishable.*

Proof. The only difference between the two hybrids is the manner in which the commitments β_1 and β_2 are computed. Let's consider the following adversary $\mathcal{A}_{\mathsf{com}}$, which internally executes the hybrid H_0 except that it does not generate the commitments β_1 and β_2 on it's own. Instead, after receiving the challenge message vectors $\boldsymbol{X}^0$ and $\boldsymbol{X}^1$ from $\mathcal{A}$, it sends two sets of strings, namely $(0^\lambda, 0^\lambda)$ and $(h(s_1), h(s_2))$ to the outside challenger where s_1 and s_2 are defined the same way as in H_1. In return, $\mathcal{A}_{\mathsf{com}}$ receives two commitments β_1, β_2 corresponding to either the first or the second set of strings. It then gives these to $\mathcal{A}$. Now, whatever bit b that $\mathcal{A}$ guesses, $\mathcal{A}_{\mathsf{com}}$ forwards the guess to the outside challenger. Clearly, $\mathcal{A}_{\mathsf{com}}$ is a polynomial time algorithm and violates the hiding property of com unless $\mathsf{H}_0 \approx_c \mathsf{H}_1$.

Lemma 2. *(*$\mathsf{H}_1 \approx_c \mathsf{H}_2$*). Assuming the semantic security of* PKE *and the strong witness indistinguishability of the proof system, the outputs of experiments* H_1 *and* H_2 *are computationally indistinguishable.*

Proof. Recall that strong witness indistinguishability asserts the following: let $\mathcal{D}_0$ and $\mathcal{D}_1$ be distributions which output an instance-witness pair for an NP-relation R and suppose that the first components of these distributions are computationally indistinguishable, i.e., $\{y : (y, w) \leftarrow \mathcal{D}_0(1^\lambda)\} \approx_c \{y : (y, w) \leftarrow \mathcal{D}_1(1^\lambda)\}$; then $\mathcal{X}_0 \approx_c \mathcal{X}_1$ where $\mathcal{X}_b : \{(\mathsf{crs}, y, \pi) : \mathsf{crs} \leftarrow \mathsf{CRSGen}(1^\lambda); (y, w) \leftarrow \mathcal{D}_b(1^\lambda); \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, y, w)\}$ for $b \in \{0, 1\}$.

Suppose that H_1 and H_2 can be distinguished with noticeable advantage δ. Note that we can visualize Hybrid H_2 as a sequence of n hybrids $\mathsf{H}_{1,0}, \ldots, \mathsf{H}_{1,n}$ where in each hybrid, the only change from the previous hybrid happens in the i^{th} challenge ciphertext CT_i. $\mathsf{H}_{1,0}$ corresponds to H_1 and $\mathsf{H}_{1,n}$ corresponds to H_2. Therefore, if H_1 and H_2 can be distinguished with advantage δ, then there exists i such that $\mathsf{H}_{1,i-1}$ and $\mathsf{H}_{1,i}$ can be distinguished with advantage δ/n where n is a polynomial in the security parameter λ. So, let's fix this i and work with these two hybrids $\mathsf{H}_{1,i-1}$ and $\mathsf{H}_{1,i}$.

Observe that both hybrids internally sample the following values in an identical manner: $\zeta = (\mathsf{pk}_1, \mathsf{pk}_2, \mathsf{pk}_3, \mathsf{pk}_4, h, \alpha, \beta_1, \beta_2, Z, c_{i,1}, c_{i,2}, d_{k_i}, k_i)$. This includes everything except $crs, c_{i,3}$ and π_i. By simple averaging, there is at least a $\delta/2n$ fraction of strings st such that the two hybrids can be distinguished with advantage at least $\delta/2n$ when $\zeta = st$. Call such a ζ to be good. Fix one such ζ, and denote the resulting hybrids by $\mathsf{H}^\zeta_{1,i-1}$ and $\mathsf{H}^\zeta_{1,i}$. Note that the hybrids have inbuilt into them all other values used to sample ζ namely : $\boldsymbol{X}^0, \boldsymbol{X}^1$ received from $\mathcal{A}$, randomness for generating the encryptions and the commitments, and the master secret key msk.

The first distribution $\mathcal{D}^{(\zeta)}_0$ is defined as follows: compute $c_{i,3} = \mathsf{PKE.Enc}(\mathsf{pk}_3, a_i; r_{i,3})$ where $a_i = x^0_i||k_i||r_{i,1}||0^{\lambda^2}||0^\lambda||0^{\lambda^2}||x^0_i||k_i||r_{i,2}||0^{\lambda^2}||0^\lambda||0^{\lambda^2}||0^\lambda$ and let statement $y = (c_{i,1}, c_{i,2}, c_{i,3}, \mathsf{pk}_1, \mathsf{pk}_2, \mathsf{pk}_3, \mathsf{pk}_4, \beta_1, \beta_2, k_i, d_{k_i}, \alpha, Z)$, witness $w = (a_i, r_{i,3}, \sigma_{k_i})$. It outputs (y, w). Note that y is identical to ζ except that h has been removed and $c_{i,3}$ has been added. Define a second distribution $\mathcal{D}^{(\zeta)}_1$ identical to $\mathcal{D}^{(\zeta)}_0$ except that instead of a_i , it uses $a^*_i = x^0_i||k_i||r_1||u_1||h(s_1)||\gamma_{i,1}||x^1_i||k_i||r_2||u_2||h(s_2)||\gamma_{i,2}||i$. Here, $\gamma_{i,1}, \gamma_{i,2}$ are the openings of the hash values in the merkle tree. That is, $\gamma_{i,1} = \mathsf{H.Open}(h, s_1, i, x^0_i||k_i)$ and $\gamma_{i,2} = \mathsf{H.Open}(h, s_2, i, x^1_i||k_i)$ where $s_1 = (x^0_1||k_1, \ldots, x^0_n||k_n)$ and $s_2 = (x^1_1||k_1, \ldots, x^1_n||k_n)$. Then, it computes $c^*_{i,3} = \mathsf{PKE.Enc}(\mathsf{pk}_3, a^*_i; r_{i,3})$, $y^* = (c_{i,1}, c_{i,2}, c^*_{i,3}, \mathsf{pk}_1, \mathsf{pk}_2, \mathsf{pk}_3, \mathsf{pk}_4, \beta_1, \beta_2, k_i, d_{k_i}, \alpha, Z)$, and $w^* = (a^*_i, r_{i,3}, \sigma_i)$. It outputs (y^*, w^*). It follows from the security of the encryption scheme that the distribution of y sampled by $\mathcal{D}^{(\zeta)}_0$ is computationally indistinguishable from y^* sampled by $\mathcal{D}^{(\zeta)}_1$, i.e., $y \approx_c y^*$. Therefore, we must have that $\mathcal{X}_0 \approx_c \mathcal{X}_1$ with respect to these distributions. We show that this is not the case unless $\mathsf{H}^\zeta_{1,i-1} \approx_c \mathsf{H}^\zeta_{1,i}$.

Consider an adversary $\mathcal{A}'$ for strong witness indistinguishability who incorporates $\mathcal{A}$ and ζ (along with sk_1 and all values for computing ζ described above),

and receives a challenge (crs, y, π) distributed according to either $\mathcal{D}_0^{(\zeta)}$ or $D_1^{(\zeta)}$; here y has one component $c_{i,3}$ that is different from ζ. The adversary $\mathcal{A}'$ uses $\mathsf{crs}, \mathsf{sk}_1$ and other values used in defining ζ to completely define PP, answer encryption key queries, generate other challenge ciphertexts and answer the function key queries and feeds it to $\mathcal{A}$. Then, it uses $(c_{i,3}, \pi)$ to define the i^{th}challenge ciphertext $\mathsf{CT}_i = (c_{i,1}, c_{i,2}, c_{i,3}, d_{k_i}, \pi, k_i)$. The adversary $\mathcal{A}'$ outputs whatever $\mathcal{A}$ outputs. We observe that the output of this adversary is distributed according to $\mathsf{H}_{1,i-1}^m$ (resp., $\mathsf{H}_{1,i}^m$) when it receives a tuple from distribution $\mathcal{X}_0$ (resp., $\mathcal{X}_1$). A randomly sampled m is good with probability at least $\delta/2n$, and therefore it follows that with probability at least $\frac{\delta^2}{4n^2}$, the strong witness indistinguishability property will be violated with non-negligible probability unless δ is negligible.

Lemma 3. *(*$\mathsf{H}_7 \approx_c \mathsf{H}_8$*). Assuming the correctness of* PKE*, that* $\mathcal{O}$ *is a public-coin differing-inputs obfuscator for for Turing machines in the class* $\mathcal{M}$*,* G *is a pseudorandom generator,* com *is a perfectly binding and (computationally) hiding commitment scheme and* H_λ *is a family of merkle hash functions, the outputs of experiments* H_7 *and* H_8 *are computationally indistinguishable.*

Proof. Suppose that the claim is false and $\mathcal{A}$'s output in H_7 is noticeably different from its output in H_8. Suppose that $\mathcal{A}$'s running time is bounded by a polynomial μ so that there are at most μ function key queries it can make. We consider a sequence of μ hybrid experiments between H_7 and H_8 such that hybrid $\mathsf{H}_{7,v}$ for $v \in [\mu]$ is as follows.

Hybrid $\mathsf{H}_{7,v}$ It is identical to H_7 except that it answers the function key queries as follows. For $j \in [\mu]$, if $j \leqslant v$, the function key corresponding to the j^{th} query, denoted by M_j , is an obfuscation of program $\mathsf{G}_{\mathsf{M}_j}$. If $j > v$, it is an obfuscation of program $\mathsf{G}'_{\mathsf{M}_j}$. We define $\mathsf{H}_{7,0}$ to be H_7 and observe that $\mathsf{H}_{7,\mu}$ is the same as H_8.

We see that if $\mathcal{A}$'s advantage in distinguishing between H_7 and H_8 is δ, then there exists a $v \in [\mu]$ such that $\mathcal{A}$'s advantage in distinguishing between $\mathsf{H}_{7,v-1}$ and $\mathsf{H}_{7,v}$ is at least δ/μ. We show that if δ is not negligible, then we can use $\mathcal{A}$ to violate the indistinguishability of the obfuscator $\mathcal{O}$. To do so, we define a sampling algorithm $\mathsf{Sam}_{\mathcal{A}}^v$ and a distinguishing algorithm $\mathcal{D}_{\mathcal{A}}^v$ and prove that $\mathsf{Sam}_{\mathcal{A}}^v$ is a public-coin differing inputs sampler outputting a pair of differing-input TMs yet $\mathcal{D}_{\mathcal{A}}^v$ can distinguish an obfuscation of left TM from that of right TM that is output by $\mathsf{Sam}_{\mathcal{A}}^v$. The description of these two algorithms is as follows:

Sampler $\mathsf{Sam}_{\mathcal{A}}^v(\rho)$:

1. Receive $(\boldsymbol{X}^0, \boldsymbol{X}^1, \mathrm{I})$ from $\mathcal{A}$.
2. Parse ρ as (crs, h, τ).
3. Proceed identically to H_7 using τ as randomness for all tasks except for sampling the hash function which is set to h, and the CRS, which is set to crs. This involves the following steps:

(a) Parse $\tau = (\tau_1, \tau_2, \tau_3, \tau_4, r_{i,1}, r_{i,2}, r_{i,3}, r_\ell, u, u_1, u_2)$ for all $i \in [n]$ and for all $\ell \in [|\mathrm{I}|]$.

(b) Use τ_1 as randomness to generate $(\mathsf{pk}_1, \mathsf{sk}_1)$, τ_2 as randomness to generate $(\mathsf{pk}_2, \mathsf{sk}_2)$ τ_3 as randomness to generate $(\mathsf{pk}_3, \mathsf{sk}_3)$ τ_4 as randomness to generate $(\mathsf{pk}_4, \mathsf{sk}_4)$.

(c) Use u as randomness to generate $\alpha = \mathsf{com}(h(s); u)$, where $s = (1||k_1, 2||k_2, \ldots, t||k_m)$ and $\{k_1, \ldots, k_m\} = \mathrm{I}$.

(d) Use u_1, u_2 as randomness to generate $\beta_1 = \mathsf{com}(h(s_1); u_1)$ and $\beta_2 = \mathsf{com}(h(s_2); u_2)$, where $s_1 = (1||x_1^0||k_1, \ldots, n||x_n^0||k_n)$ and $s_2 = (1||x_1^1||k_1, \ldots, n||x_n^1||k_n)$.

(e) Define Z to be a uniform random string of length 2λ. Define the public parameters $\mathsf{PP} = (\mathsf{crs}, \mathsf{pk}_1, \mathsf{pk}_2, \mathsf{pk}_3, \mathsf{pk}_4, h, \alpha, \beta_1, \beta_2, Z)$. Send PP to $\mathcal{A}$.

(f) For all $k_i \in \mathrm{I}$, to generate the i^{th} encryption key EK_{k_i}, compute $b_{k_i} = z||h(s)||\gamma_i||u_1||i$ and $d_{k_i} = \mathsf{PKE.Enc}(\mathsf{pk}_4, b_{k_i}; r_i)$. Using witness $w_{k_i} = (b_{k_i}, r_i)$, compute proof σ_{k_i} using the AND of statements 1 and 2(b) in the relation R_1.
Send the encryption key EK_{k_i} for all $k_i \in \mathrm{I}$ to $\mathcal{A}$.

(g) For all $i \in [n]$, we generate the i^{th} challenge ciphertext in the following manner. We use $r_{i,1}$ and $r_{i,2}$ as randomness to generate $c_{i,1} = \mathsf{PKE.Enc}(\mathsf{pk}_1, x_i^0||k_i; r_{i,1})$ and $c_{i,2} = \mathsf{PKE.Enc}(\mathsf{pk}_2, x_i^1||k_i; r_{i,2})$. Use $a_i = x_i^0||k_i||r_{i,1}||u_1||h(s_1)||\gamma_{i,1}||x_i^1||k_i||r_{i,2}||u_2||h(s_2)||\gamma_{i,2}||i$ where $\gamma_{i,1}, \gamma_{i,2}$ are the openings for $h(s_1)$ and $h(s_2)$ w.r.t. $x_i^0||k_i$ and $x_i^1||k_i$ respectively. That is, $\gamma_{i,1} = \mathsf{H.Open}(h, s_1, i, x_i^0||k_i)$ and $\gamma_{i,2} = \mathsf{H.Open}(h, s_2, i, x_i^1||k_i)$. Compute $c_{i,3} = \mathsf{PKE.Enc}(\mathsf{pk}_3, a_i; r_{i,3})$. Then, use witness $w_i = (a_i, r_{i,3}, \sigma_{k_i})$ to compute proof π_i using the AND of statements 1 and 2(b) in the relation R_2. The i^{th} challenge ciphertext is $(c_{i,1}, c_{i,2}, c_{i,3}, d_{k_i}, \pi_i, k_i)$.
Send all the challenge ciphertexts to $\mathcal{A}$.

(h) Answer the function key queries of $\mathcal{A}$ as follows. For all queries M_j, until $j < v$, send an obfuscation of $\mathsf{G}_{\mathsf{M}_j}$.

(i) Upon receiving the v^{th} function key query M_v, output $(\tilde{\mathsf{M}}_0, \tilde{\mathsf{M}}_1)$ and halt, where :

$$\tilde{\mathsf{M}}_0 = \mathsf{G}_{\mathsf{M}_v}, \qquad \tilde{\mathsf{M}}_1 = \mathsf{G}'_{\mathsf{M}_v}.$$

Distinguisher $\mathcal{D}^v_{\mathcal{A}}(\rho, \mathsf{M}')$: on input a random tape ρ and an obfuscated TM M', the distinguisher simply executes all steps of the sampler $\mathsf{Sam}^v_{\mathcal{A}}(\rho)$, answering function keys for all $j < v$ as described above. The distinguisher, however, does not halt when the v^{th} query is sent, and continues the execution of $\mathcal{A}$ answering function key queries for M_j as follows :

- if $j = v$, send M' (which is an obfuscation of either $\tilde{\mathsf{M}}_0$ or $\tilde{\mathsf{M}}_1$).
- if $j > v$, send an obfuscation of $\mathsf{G}'_{\mathsf{M}_j}$.

The distinguisher outputs whatever $\mathcal{A}$ outputs.

We can see that if M' is an obfuscation of $\tilde{\mathsf{M}}_0$, the output of $\mathcal{D}^v_{\mathcal{A}}(\rho, \mathsf{M}')$ is identical to $\mathcal{A}$'s output in $\mathsf{H}_{7,k-1}$ and if M' is an obfuscation of $\tilde{\mathsf{M}}_1$, it is identical

to $\mathcal{A}$'s output in $\mathsf{H}_{7,k}$. We have that $\mathcal{D}^v_{\mathcal{A}}(\rho, \mathsf{M}')$ distinguishes $\mathsf{H}_{7,k-1}$ and $\mathsf{H}_{7,k}$ with at least δ/μ advantage.

All that remains to prove now is that $\mathsf{Sam}^v_{\mathcal{A}}(\rho)$ is a public-coin differing-inputs sampler.

Theorem 3. *$\mathsf{Sam}^v_{\mathcal{A}}(\rho)$ is a public-coin differing inputs sampler.*

Proof. We show that if there exists an adversary B who can find differing-inputs to the pair of TMs sampled by $\mathsf{Sam}^v_{\mathcal{A}}(\rho)$ with noticeable probability, we can use B and $\mathsf{Sam}^v_{\mathcal{A}}(\rho)$ to construct an efficient algorithm $\mathsf{CollFinder}_{\mathsf{B},\mathsf{Sam}^v_{\mathcal{A}}(\rho)}$ which finds collisions in h with noticeable probability.

$\mathsf{CollFinder}_{\mathsf{B},\mathsf{Sam}^v_{\mathcal{A}}(\rho)}(h)$:
On input a random hash function $h \leftarrow H_\lambda$, the algorithm first samples uniformly random strings (crs, τ) to define a random tape $\rho = (\mathsf{crs}, h, \tau)$. Then, it samples $(\tilde{\mathsf{M}}_0, \tilde{\mathsf{M}}_1) \leftarrow \mathsf{Sam}^v_{\mathcal{A}}(\rho)$ and computes $e^* \leftarrow \mathsf{B}(\rho)$ e^* is the differing input and corresponds to a set of ciphertexts. Let $e^* = (e^*_1, \ldots, e^*_\ell)$ where each $e^*_j = (e^*_{j,1}, e^*_{j,2}, e^*_{j,3}, d^*_{k^*_j}, \pi^*_j, k^*_j)$ for $j \in [\ell]$. For each j, if π^*_j is a valid proof, compute $a^*_j = \mathsf{PKE.Dec}(\mathsf{sk}_3, e^*_{j,3})$ and let $a^*_j = x^*_{j,1}||k^*_{j,1}||r^*_{j,1}||u_1||\mathsf{hv}^*_1||\gamma^*_{j,1}||x^*_{j,2}||k^*_{j,2}||r^*_{j,2}||u_2||\mathsf{hv}^*_2||\gamma^*_{j,2}||t^*$. Let $(\boldsymbol{X}^0, \boldsymbol{X}^1)$ be the challenge message vectors output by $\mathcal{A}$ initially. Let $\boldsymbol{X}^0 = \{(x^0_1, k_1), \ldots, (x^0_n, k_n)\}$ and $\boldsymbol{X}^1 = \{(x^1_1, k_1), \ldots, (x^1_n, k_n)\}$. Define $s_1 = (x^0_1||k_1, \ldots, x^0_n||k_n)$ and $s_2 = (x^1_1||k_1, \ldots, x^1_n||k_n)$ Let the encryption key queries be $\mathrm{I} = \{k_1, \ldots, k_t\}$. Define $s = (k_1, \ldots, k_t)$. If $h(s_1) = h(s_2)$, output (s_1, s_2) as collisions to the hash function.

Claim. For all $j \in [\ell]$, π^*_j is a valid proof.

Proof. Since e^* is a differing input, $\tilde{\mathsf{M}}_0(e^*) \neq \tilde{\mathsf{M}}_1(e^*)$. Now, suppose for some $j \in [\ell]$, π^*_j was not a valid proof. Then, both $\tilde{\mathsf{M}}_0$ and $\tilde{\mathsf{M}}_1$ would output $\perp$ on input e^* which means that e^* is not a differing input.

Condition A : A ciphertext $C = (c_1, c_2, c_3, d_k, \pi, k)$ for which π is valid satisfies condition A with respect to challenge message vectors $(\boldsymbol{X}^0, \boldsymbol{X}^1)$ and encryption key queries I iff

1. c_1 and c_2 encrypt the same message and $k \in \mathrm{I}$ (OR)
2. $\exists i \in [n]$ such that $\{(x_1||k_1), (x_2||k_2)\} = \{(x^0_i||k_i), (x^1_i||k_i)\}$, where $x_1||k_1 = \mathsf{PKE.Dec}(\mathsf{sk}_1, c_1)$ and $x_2||k_2 = \mathsf{PKE.Dec}(\mathsf{sk}_2, c_2)$.

Claim. For every $j \in [\ell]$, if e^*_j satisfies condition A, then e is not a differing input.

Proof. Suppose the above two conditions are true for every $j \in [\ell]$. Then, from the definition of I-compatibility of challenge message vectors $(\boldsymbol{X^0}, \boldsymbol{X}^1)$ and function query M_v, we see that $\tilde{\mathsf{M}}_0(e^*) = \tilde{\mathsf{M}}_1(e^*)$ which means that e^* is not a differing input.

Therefore, since we have assumed that e^* is a differing input, there exists $j \in [\ell]$ such that e^*_j does not satisfy condition A.

Claim. If there exists $j \in [\ell]$ such that e^*_j does not satisfy condition A, then we can find a collision in the hash function h.

Proof. Let's fix $j \in [\ell]$ such that e^*_j does not satisfy condition A. Since π^*_j is a valid proof, by the soundness of the strong witness indistinguishable proof system, one of the following two cases must hold :

- **case 1:** π^*_j was proved using statements 1 and 2(a) of relation R_2.

  Now, since e^*_j does not satisfy condition A, it doesn't satisfy condition A(1) as well. Therefore, either $e^*_{j,1}$ and $e^*_{j,2}$ encrypt different messages or $k^*_j \notin \mathrm{I}$. If $e^*_{j,1}$ and $e^*_{j,2}$ encrypt different messages, statement 2(a) would clearly be false and π^*_j would not be valid. However, we already proved that π^*_j is valid. Therefore, it must be the case that $k^*_j \notin \mathrm{I}$.
  Since 2(a) is true in R_2, we have $\mathsf{Verify}(\mathsf{crs}, st_{k^*_j}, \sigma_{k^*_j}) = 1$ where $st_{k^*_j} = (d_{k^*_j}, k^*_j, \mathsf{pk}_4, \alpha, Z)$ and $\sigma_{k^*_j}$ is a proof that $st_{k^*_j} \in L_1$. Further, since Z is a uniform random string, $Z \neq \mathsf{G}(z)$ for any z except with negligible probability. As a result, $\sigma_{k^*_j}$ must be proved using statements 1 and 2(b) in relation R_1. Therefore, there exists $\mathsf{hv}^*, \gamma^*, t^*$ such that $\mathsf{H.Verify}(h, \mathsf{hv}^*, k^*_j, \gamma^*, t^*) = 1$ and $\mathsf{com}(\mathsf{hv}^*; u) = \alpha$. Since the commitment scheme is perfectly binding, $\mathsf{hv}^* = h(s)$. We know that $s = (k_1, \ldots, k_t)$. Therefore, $s[t^*] \neq k^*_j$. Thus, there exists γ^*, t^* such that $\mathsf{H.Verify}(h, h(s), k^*_j, \gamma^*, t^*) = 1$ and $s[t^*] \neq k^*_j$. By definition 6, we have found a collision in the hash function h.

- **case 2:** π^*_j was proved using statements 1 and 2(b) of relation R_2.

  Since e^*_j does not satisfy condition A, it doesn't satisfy condition A(2) as well. Therefore, $\forall i \in [n]\ \{(x^*_{j,1}||k^*_{j,1}), (x^*_{j,2}||k^*_{j,2})\} \neq \{(x^0_i||k_i), (x^1_i||k_i)\}$. Since π^*_j was proved using 2(b), $\exists \mathsf{hv}^*_1, \mathsf{hv}^*_2, \gamma^*_1, \gamma^*_2, t^*$ such that 2(b)(ii) and 2(b)(iii) are true. Without loss of generality, let's say that the first of the two conditions in 2(b)(ii) is true and the second of the two conditions in 2(b)(iii) is true. That is, $\mathsf{H.Verify}(h, \mathsf{hv}^*_1, x^*_{j,1}||k^*_{j,1}, \gamma^*_1, t^*) = 1$, $\beta_1 = \mathsf{com}(\mathsf{hv}^*_1; u_1)$ and $\mathsf{H.Verify}(h, \mathsf{hv}^*_2, x^*_{j,2}||k^*_{j,2}, \gamma^*_2, t^*) = 1$, $\beta_2 = \mathsf{com}(\mathsf{hv}^*_2; u_2)$. Since the commitment scheme is perfectly binding, $\mathsf{hv}^*_1 = h(s_1)$ and $\mathsf{hv}^*_2 = h(s_2)$. We know that $\{(x^*_{j,1}||k^*_{j,1}), (x^*_{j,2}||k^*_{j,2})\} \neq \{(x^0_{t^*}||k_{t^*}), (x^1_{t^*}||k_{t^*})\}$. Without loss of generality, let's say $(x^*_{j,1}||k^*_{j,1}) \neq (x^0_{t^*}||k_{t^*})$. Since $s_1 = (x^0_1||k_1, \ldots, x^0_n||k_n)$, we have $s_1[t^*] \neq (x^*_{j,1}||k^*_{j,1})$. Thus, there exists γ^*_1, t^* such that $s_1[t^*] \neq x^*_{j,1}||k^*_{j,1}$ and $\mathsf{H.Verify}(h, h(s_1), x^*_{j,1}||k^*_{j,1}, \gamma^*_1, t^*) = 1$. By definition 6, we have found a collision in the hash function h.

References

1. Ananth, P., Boneh, D., Garg, S., Sahai, A., Zhandry, M.: Differing-inputs obfuscation and applications. (2013) 3

2. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K.: On the (im) possibility of obfuscating programs. In: Advances in cryptology—CRYPTO 2001. pp. 1–18. Springer (2001) 2, 3
3. Boneh, D., Sahai, A., Waters, B.: Functional encryption: Definitions and challenges. In: Theory of Cryptography, pp. 253–273. Springer (2011) 2
4. Boneh, D., Waters, B.: Conjunctive, subset, and range queries on encrypted data. In: Theory of cryptography, pp. 535–554. Springer (2007) 2
5. Boyle, E., Chung, K.M., Pass, R.: On extractability (aka differing-inputs) obfuscation. TCC (2014) 3
6. Boyle, E., Pass, R.: Limits of extractability assumptions with distributional auxiliary input. IACR Cryptology ePrint Archive 2013, 703 (2013), http://eprint.iacr.org/2013/703 3
7. Feige, U., Lapidot, D., Shamir, A.: Multiple noninteractive zero knowledge proofs under general assumptions. SIAM Journal on Computing 29(1), 1–28 (1999) 9
8. Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. In: Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on. pp. 40–49. IEEE (2013) 2
9. Garg, S., Gentry, C., Halevi, S., Wichs, D.: On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. In: Advances in Cryptology–CRYPTO 2014, pp. 518–535. Springer (2014) 3
10. Goldwasser, S., Gordon, S.D., Goyal, V., Jain, A., Katz, J., Liu, F.H., Sahai, A., Shi, E., Zhou, H.S.: Multi-input functional encryption. In: Advances in Cryptology–EUROCRYPT 2014, pp. 578–602. Springer (2014) 2, 3, 4, 5, 6
11. Goldwasser, S., Goyal, V., Jain, A., Sahai, A.: Multi-input functional encryption. Cryptology ePrint Archive, Report 2013/727 (2013) 2, 10, 12
12. Gordon, S.D., Katz, J., Liu, F.H., Shi, E., Zhou, H.S.: Multi-input functional encryption. IACR Cryptology ePrint Archive 2013, 774 (2013) 2, 10, 12
13. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: Proceedings of the 13th ACM conference on Computer and communications security. pp. 89–98. Acm (2006) 2
14. Halevi, S., Lindell, Y., Pinkas, B.: Secure computation on the web: Computing without simultaneous interaction. In: CRYPTO. pp. 132–150 (2011) 2, 4
15. Ishai, Y., Pandey, O., Sahai, A.: Public-coin differing-inputs obfuscation and its applications. In: TCC. pp. 668–697 (2015) 7
16. Katz, J., Sahai, A., Waters, B.: Predicate encryption supporting disjunctions, polynomial equations, and inner products. In: Advances in Cryptology–EUROCRYPT 2008, pp. 146–162. Springer (2008) 2
17. Lewko, A., Okamoto, T., Sahai, A., Takashima, K., Waters, B.: Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In: Advances in Cryptology–EUROCRYPT 2010, pp. 62–91. Springer (2010) 2
18. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Advances in Cryptology—CRYPTO'87. pp. 369–378. Springer (1988) 9
19. O'Neill, A.: Definitional issues in functional encryption. IACR Cryptology ePrint Archive 2010, 556 (2010) 2
20. Pandey, O., Prabhakaran, M., Sahai, A.: Obfuscation-based non-black-box simulation and four message concurrent zero knowledge for np. IACR Cryptology ePrint Archive 2013, 754 (2013) 3
21. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: Advances in Cryptology–EUROCRYPT 2005, pp. 457–473. Springer (2005) 2