Ad Hoc Multi-Input Functional Encryption

Shweta Agrawal
Dept. of Computer Science and Engineering, IIT Madras, Chennai, India
shweta@iitm.ac.in

Michael Clear
Dept. of Computer Science, Georgetown University, Washington, DC, USA
clrmic2@gmail.com

Ophir Frieder
Dept. of Computer Science, Georgetown University, Washington, DC, USA
ophir@ir.cs.georgetown.edu

Sanjam Garg
Dept. of EECS, University of California at Berkeley, CA, USA
sanjamg@berkeley.edu

Adam O'Neill
Dept. of Computer Science, University of Massachusetts, Amherst, MA, USA
amoneill@gmail.com

Justin Thaler
Dept. of Computer Science, Georgetown University, Washington, DC, USA
Justin.Thaler@georgetown.edu

Abstract

Consider *sources* that supply sensitive data to an *aggregator*. Standard encryption only hides the data from eavesdroppers, but using specialized encryption one can hope to hide the data (to the extent possible) from the aggregator itself. For flexibility and security, we envision schemes that allow sources to supply encrypted data, such that at any point a dynamically-chosen subset of sources can allow an agreed-upon joint function of their data to be computed by the aggregator. A primitive called multi-input functional encryption (MIFE), due to Goldwasser et al. (EUROCRYPT 2014), comes close, but has two main limitations:

- it requires trust in a third party, who is able to decrypt all the data, and
- it requires function arity to be fixed at setup time and to be equal to the number of parties.

To drop these limitations, we introduce a new notion of *ad hoc* MIFE. In our setting, each source generates its own public key and issues individual, function-specific secret keys to an aggregator. For successful decryption, an aggregator must obtain a separate key from each source whose ciphertext is being computed upon. The aggregator could obtain multiple such secret-keys from a user corresponding to functions of varying arity. For this primitive, we obtain the following results:

- We show that standard MIFE for general functions can be bootstrapped to ad hoc MIFE for *free*, i.e. without making any additional assumption.
- We provide a direct construction of ad hoc MIFE for the inner product functionality based on the Learning with Errors (LWE) assumption. This yields the first construction of this natural primitive based on a standard assumption.

At a technical level, our results are obtained by combining standard MIFE schemes and *two-round* secure multiparty computation (MPC) protocols in novel ways highlighting an interesting interplay between MIFE and two-round MPC.

2012 ACM Subject Classification Security and privacy → Cryptography; Security and privacy → Public key (asymmetric) techniques; Security and privacy → Mathematical foundations of cryptography

Keywords and phrases Multi-Input Functional Encryption

Digital Object Identifier 10.4230/LIPIcs.ITCS.2020.40

11th Innovations in Theoretical Computer Science Conference (ITCS 2020).
Editor: Thomas Vidick; Article No. 40; pp. 40:1–40:41
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Related Version A full version of the paper is available at `https://eprint.iacr.org/2019/356`.

Funding *Sanjam Garg*: Research supported in part from DARPA/ARL SAFEWARE Award W911NF15C0210, AFOSR Award FA9550- 15-1-0274, AFOSR YIP Award, DARPA and SPAWAR under contract N66001-15-C-4065, a Hellman Award and research grants by the Okawa Foundation, Visa Inc., and Center for Long-Term Cybersecurity (CLTC, UC Berkeley). The views expressed are those of the author and do not reflect the official policy or position of the funding agencies.

Acknowledgements We thank Romain Gay for pointing out a subtlety in the security definition of (standard) MIFE for inner products, which necessitated a minor adjustment in our ad hoc MIFE construction.

1 Introduction

In modern society, there is an inherent need for external entities to aggregate and analyze sensitive data from a variety of sources. A few prominent examples are:

- To track diseases, disease control centers would like hospital patients' medical information.
- To determine medication efficacy for a given subpopulation, pharmaceutical companies would like patients' genomic information.
- To provide targeted advertising to consumers, corporations would like buyers' demographic information.

However, this release of sensitive data to external entities is unsettling, as these entities must now be trusted to preserve the confidentiality of the released data. We would like to avoid the need for this trust and believe that specialized encryption schemes will be an important tool for doing so. At a high level, we would like schemes that allow users to encrypt their data before transferring them to an external entity, such that only certain user-specified joint functions of the data are revealed to the entity holding it. We would like this security guarantee to be supported in a flexible way, allowing joint functions of the data to be revealed by any dynamically-chosen subset of users that permit it.

A primitive that comes close, due to Goldwasser et al. [45], is *multi-input functional encryption* (MIFE). To understand MIFE, we first recall the simpler notion of functional encryption (FE) [20]. Just as in traditional encryption, in functional encryption ciphertexts can be generated with an encryption key. However, each *decryption* key is associated with a function f, and decryption of an encryption of m using this key results in not m but $f(m)$. Intuitively, security requires that nothing more than $f(m)$ can be learned from the encryption of m and the decryption key for f. In MIFE, decryption keys allow computing *joint* functions of (possibly) different plaintexts underlying multiple ciphertexts. That is, decryption takes a key for a function f and ciphertexts $c_1, \dots c_n$ encrypting $m_1, \dots, m_n$, and outputs $f(m_1, \dots, m_n)$.

However, MIFE has an important drawback: encryption and decryption keys are generated via a global setup procedure run by an external entity usually called the *key authority*. This begs the question of whether putting trust in the key authority is really better than putting trust in the external entities that aggregate and analyze the data in the first place. A similar point was made by Rogaway about identity-based encryption (IBE) [60]. Indeed, removing this in the case of IBE (and other settings) has been an active area of investigation, *e.g.* see [18, 49, 35]. We contend that for MIFE (and indeed FE) the concern is heightened, as the authority can not only decrypt all the data but is also the one in charge of which functions of the data other external entities can compute. Hence, MIFE does not allow users to enforce their own privacy policies.

Additionally, from a flexibility standpoint, MIFE is limited in that it fixes the number of senders and function arity at setup time. This does not support a dynamic setting in which users can join or leave. Progress on removing this limitation was made by Badrinarayanan et al. [10], who introduced a notion of MIFE for unbounded arity functions. However, their notion does not allow any subset of users to reveal a joint function of their data to an external entity without coordination from all other users, and moreover relies on strong "knowledge type" assumptions. Another closely related notion to ours is that of *decentralized multi-client FE*, recently introduced by Chotard et al. [27]. While this work shares some of the high level motivation of the present work, the two primitives are very different. Please see Section 1.3.3 for a comparison.

1.1 Our Notion: *Ad Hoc* MIFE

To address the above limitations, we introduce a new notion of *ad hoc* MIFE. In ad hoc MIFE, each source (aka. sender or user) will run a *local* setup procedure to generate some public parameters as well as a private encryption key. (One can also consider a public-key setting, but this puts limits on achievable security and we do not do so in this work.[1]) Each source publishes their public parameters and encrypts using their private key. These ciphertexts can be sent to an aggregator (aka. decryptor). Furthermore, using their private keys, sources can issue "partial decryption keys" to an aggregator. Each partial decryption key is associated with an ℓ-ary function f for some ℓ and is generated using the public keys of $\ell - 1$ other (dynamically chosen) sources. If these other $\ell - 1$ sources also issue "matching" partial decryption keys for f to this aggregator, it can decrypt any ℓ ciphertexts $c_1, \ldots, c_\ell$, each produced by the corresponding source, to $f(m_1, \ldots, m_\ell)$ where $m_1, \ldots, m_\ell$ are the plaintexts. One can also consider restricted versions of the above notion, that bound the number of users or fix ℓ (or both). In particular, taking the number of users equal to ℓ gives a version of MIFE that still drops the global setup procedure but lacks the dynamic aspect. Finally, we also consider the restricted notion of *bounded* ad hoc MIFE, where we place a bound on the number of "partial decryption keys" a user is allowed to issue. Intuitively, for security, we require that an aggregator learns only the functions of the data for which it has been given all of the matching partial decryption keys.

Note that while we assume each source can obtain the authentic public parameters of other sources with whom it wants to allow joint functions of the data to be computed, there is no other prior coordination between users (this is one of the main advantages of our notion over [27]). In particular, there is no external entity that generates public parameters or keys. In some of our constructions, we work in the common reference string (CRS) model, but note that this is still much weaker than having an authority who can decrypt all the data.

1.2 Our Results

Our results may be summarized as follows:

- *Feasibility result for general functions*: First, we show that standard MIFE for general functions can be bootstrapped to ad hoc MIFE for general functions for *free*. More specifically, we show that ad hoc MIFE for any functionality is implied by standard MIFE for that functionality combined in a novel way with general FE and a special type of *two-round* secure multiparty computation (MPC) protocol. The latter two are implied by standard MIFE for general functions.

[1] In more detail, in the public-key setting a decryptor could launch an attack where it replaces one user's input with various values to determine information about the input of another user.

Table 1 Our new constructions of ad hoc MIFE.

Functionality	Assumptions	Security	CRS?	Section
General	std. MIFE	Semi-honest	No	4
General	std. MIFE	Malicious	Yes	4
Inner Product	LWE	Semi-honest	Yes	5
Inner Product	LWE	Malicious	Yes	5
Inner Product (Bounded)	DDH, LWE, DCR	Semi-honest	No	5
Inner Product (Bounded)	DDH, LWE, DCR	Malicious	Yes	5

While very general, the result leaves open the goal of obtaining ad hoc MIFE under standard assumptions. In general, this is challenging as standard MIFE is already known to be equivalent to indistinguishability obfuscation [33, 8, 17], which is a central open problem in cryptography. In fact, some negative evidence about the hardness of obtaining such constructions has also been provided [37, 38]. Thus, with the goal of moving towards using standard assumptions, we consider the task of ad hoc MIFE for special but natural functionalities.

- *Constructions for Inner Products from Standard Assumptions*: We provide a construction of ad hoc MIFE for the *inner product* functionality from standard assumptions, namely LWE.[2] Introduced by Abdalla et al. [1] in the single-input setting, this functionality has applications in data mining and information retrieval. Our result is obtained via a general paradigm for constructing ad-hoc MIFE schemes from standard MIFE schemes satisfying certain natural properties; or, what we call "ad hoc friendly" standard MIFE schemes. We show that certain constructions of standard MIFE scheme for inner products from the literature [3, 2] based on standard assumptions (DDH, LWE, or DCR) already satisfy these properties. Additionally, we use a specific two-round MPC protocol [57] that can also be obtained via LWE. We note that by using a two-round MPC protocol from any two-round OT protocol [41, 42, 15] here, we also obtain results for the case of *bounded* ad hoc MIFE for inner products – namely, we get bounded ad hoc MIFE for inner products from DDH, LWE and DCR as well.[3] We remark that since our general construction (first result) already relies on general MIFE for circuits, there is no advantage to mitigating assumptions for the two-round MPC protocol in that setting.

  We emphasize that our transformation is general. Thus, our transformation can be used to upgrade the security of any "ad hoc friendly" standard MIFE for a given functionality to ad hoc MIFE for the same functionality. This result might also be useful in obtaining future constructions of ad hoc MIFE. Furthermore, the modularity of this approach allows for simplifications in our constructions.

We tabulate our results in Table 1 and provide explanation of which MPC protocol is needed for each of the results in Section 1.3.

[2] We stress that this functionality outputs inner products in the clear and is therefore a different type of functionality than that of Katz et al. in [54], which tests if an inner product is zero or not.

[3] Note that semi-honest constructions of two-round OT are known under each of these assumptions.

1.3 Technical Overview

In this section, we describe at a high level the challenges involved in constructing ad hoc MIFE and our techniques for overcoming them.

1.3.1 Ad Hoc MIFE for Arbitrary Functions.

Standard MIFE and ad hoc MIFE can be seen as secure multiparty computation (MPC) protocols with a particular allowable interaction pattern and certain additional reuse capabilities.[4] To begin, let us consider the interaction pattern followed by standard MIFE. In standard MIFE, there is a *trusted* global setup which receives as input the number of parties ℓ, and outputs a public key and a set of ℓ encryption keys. Additionally, global setup on input a function f generates the decryption key DK_f. Of these, the public parameters are broadcast to all users and encryption key EK_i is provided to encryptor i, for $i \in [\ell]$. The encryptors then compute their ciphertexts $\mathsf{CT}(m_i)$ and send these to the aggregator who may now compute the function output $f(m_1 \ldots, m_\ell)$ using the decryption/function key DK_f. Finally, the system supports arbitrary number of decryption keys and ciphertexts. As explained in Section 1.1, in ad hoc MIFE, we seek to eliminate the trusted global procedure as well as support dynamic choice of parties involved in any function computation.

1.3.1.1 Challenges Involved

An approach to eliminating the trusted setup from standard MIFE is to use MPC to replace the global setup. However, naively computing the setup procedure using MPC would introduce interaction between the parties, which the syntax of MIFE does not allow. Moreover, this (interactive) procedure would need to be rerun each time a function key is required to be generated. Using two round MPC, one may hope to overcome the barrier of interaction using the following natural idea: let parties perform a two-round MPC to perform the setup and key generation for a standard MIFE. In more detail, parties in the first round could publish as their public parameters the first round MPC messages with their secret randomness as input. Given the first round messages, parties could send the second round MPC message to the aggregator, who could use it to compute the function key. However, this approach does not suffice since:

1. MIFE requires that the public parameters be published only once whereas the above template requires publishing fresh public parameters for each function key.
2. Even more fundamentally, the above approach precludes users from being able to encrypt, as their encryption keys are not available given just the first round MPC message.

In particular, the above approach does not decouple ciphertexts and functions as in traditional MIFE, which leads to the limitation that an evaluator cannot evaluate the same function on multiple inputs chosen by the parties, nor evaluate other functions on the same set of inputs. Additionally, the problem is made challenging by the fact that in MIFE an aggregator might obtain arbitrary number of secret keys and an encryptor might generate arbitrary number of ciphertexts.

[4] Recall that MPC allows a set of parties to compute a joint function of their inputs without revealing anything else but in general allows these parties to freely interact (although restrictions may apply in special cases).

1.3.1.2 Overcoming the first barrier: Function re-runnable two-round MPC

In order to mitigate the first problem above, we require that the first round MPC message be sent only once, and reused for all subsequent second round messages thus providing re-usability/re-runnability for secret key generation. Towards achieving this re-usability, an idea is to use function rerunnable two-round MPC protocols, where the same first round message can be reused for multiple functions in the second round. As we will see, certain existing two-round MPC protocols satisfy this requirement (see later), but this still does not solve the problem. This is because in adhoc MIFE, we additionally need that for the MPC protocol, the function or even its arity are not known at the time the first round message is sent. We overcome this hurdle by using "function delayed" protocols, which permit the choice of function to be delayed to the second round of the protocol. Together, these special protocols may be used to overcome the first barrier outlined above.

1.3.1.3 Overcoming the second barrier: Delaying Encryption

In order to overcome the second barrier, we allow the encryptor to delay the encryption process until the encryption keys are known (in similar spirit as [13, 30]). In more detail, we will have each source independently run the setup algorithm of a single input FE scheme, denoted as FE and compute the first round message of an MPC protocol using the FE master key as input. This message is published as part of the public key and made available to all other sources. Additionally, each source provides an encryption of its input m_i using the algorithm $\mathsf{FE.Encrypt}$.

The sources may choose the function f to be computed and the group that will participate in the computation dynamically. At this point, each source independently executes the partial key generation algorithm as follows: it generates the second round message of an MPC protocol for a suitable f dependent functionality $\mathsf{GenKeys}_f$ and sends this to the aggregator. Intuitively, the functionality $\mathsf{GenKeys}_f$ has the circuit f hard-coded in it, and enables the aggregator to compute the output.

However, recall that the inputs to the $\mathsf{GenKeys}$ functionality are not the messages on which the computation must be performed, but rather the FE master keys generated independently by each player. To proceed, the functionality instead uses the FE master keys to compute FE *function* keys for a re-encryption procedure, which *translates* FE ciphertexts to MIFE ciphertexts for a freshly generated (standard) MIFE scheme. During this time, the arity of the function f is known, so a suitable standard MIFE scheme may be instantiated. It further outputs an MIFE function key for the function f.

We are almost done: $\mathsf{GenKeys}_f$ runs the setup procedure for a suitable fixed arity standard MIFE scheme, computes FE function keys for each party for the re-encryption functionality, computes the MIFE function key for f and outputs these. The aggregator uses the FE keys together with the FE ciphertexts provided by each encryptor to translate $\mathsf{FE.Enc}(m_i)$ to $c_i = \mathsf{MIFE.Enc}(m_i)$ and then runs the MIFE decryption procedure to obtain $f(m_1, \ldots, m_\ell)$.

Put together, we resolved all difficulties by carefully nesting a a multi-input FE scheme MIFE, within the single input FE scheme FE, which in turn is nested within a re-runnable two-round MPC protocol MPC.

One final problem remains: the adversary could get some partial information from an "incomplete" set of partial decryption keys for some function. This is because standard MPC makes no guarantee when an honest party does not send their final message. We solve this problem by masking the output of MPC by pseudorandom values generated for each party. The partial decryption keys contains the respective user's pseudorandom masks so that only a complete set of user keys can be used to unmask the output.

While security appears to follow intuitively from the security of MPC, FE and MIFE, the proof must contend with several technical hurdles as we are forced to deal with indistinguishability style security of FE, MIFE (simulation security for these primitives is known to be impossible [20, 6]). We argue security via a careful sequence of hybrids, please see Section 4 for details.

1.3.1.4 Instantiating MPC

We now discuss possible instantiations of MPC to fit the above template. Depending on the properties of the underlying two-round MPC protocol, we obtain different properties of the resulting ad hoc MIFE scheme. In both the semi-honest (passive decryptor) and the malicious (active decryptor) settings, the most general function-rerunnable, two-round MPC protocols without CRS can be constructed [32, 48] from indistinguishability obfuscation [33], which itself can be constructed from multi-input functional encryption [8]. Furthermore, as already noted in [40], we remark that in the semi-honest setting the construction of [32] can actually be instantiated in the plain model. This is based on the observation that the CRS in the protocol of [32] was only needed for the computation in the second round. Thus semi-honest parties could obtain a CRS by just performing a one-round coin flipping in the first round. This yields our first result: we get ad hoc MIFE for general functions from standard MIFE for general functions. In the semi-honest setting, the ad hoc MIFE construction is in the plain model. On the other hand, in the malicious setting, these protocols work in the common reference string (CRS) model.

Alternatively, function-rerunnable two-round MPC in the common reference string (CRS) model can be constructed [28, 57, 24, 59] from learning-with-errors (LWE). This yields ad hoc MIFE from LWE and standard MIFE in the CRS model (either semi-honest or malicious). While bounded two-round MPC in the CRS model can be constructed from bilinear maps [41] and even two-round oblivious transfer [15, 42, 39] or information theoretically [9, 36], these constructions are *not* function-rerunnable so do not suffice for our general construction. We note that these constructions would suffice for obtaining bounded ad hoc MIFE, where a user issues only a bounded number of partial decryption keys and maintains *state* across key issues. However, since in our general result we anyway require the minimum assumption of FE/MIFE, instantiating MPC from weaker assumptions does not yield any benefits, and we do not discuss this further. Our results are highlighted in Table 1.

1.3.2 Ad Hoc MIFE for Inner Products.

While our construction of ad hoc MIFE above applies to arbitrary functionalities, it requires use of standard MIFE for general functions. Unfortunately, as noted above, standard MIFE implies indistinguishability obfuscation. Hence, there is limited hope of basing it on standard assumptions. Additionally, our general transformation uses an FE scheme for a potentially complicated re-encryption functionality and also requires general-purpose, function-rerunnable two-round MPC for computing a complex functionality. These aspects limit the practical applicability of our general result.

Next, we describe a paradigm for constructing ad-hoc MIFE schemes from standard MIFE schemes that are “ad hoc friendly” and a (hopefully simple) two-round MPC protocol. This paradigm significantly simplifies our general construction and provides a way for basing it on standard assumptions. We then show that the standard MIFE scheme for inner products [3, 2], which may be based on DDH, LWE or DCR, is ad hoc friendly and the corresponding two-round MPC protocol is only required to compute inner-products, thus obtaining an efficient ad hoc MIFE scheme for inner products.

More formally, in the inner product functionality a decryption key corresponds to a concatenated vector $\mathbf{y} = (\mathbf{y_1} \parallel \cdots \parallel \mathbf{y_n})$ where $\mathbf{y_i} \in \mathbb{Z}_q^m$, and a ciphertext encrypts a vector $\mathbf{x_i} \in \mathbb{Z}_q^m$. The desired result of decryption is $\sum_{i=1}^{n} \langle \mathbf{x_i}, \mathbf{y_i} \rangle$. Importantly, the decryptor should not learn the partial sums $\langle \mathbf{x_i}, \mathbf{y_i} \rangle$. The inner product functionality has applications in data mining and information retrieval [1, 3]. We use constructions of standard MIFE for inner product by [3, 2].

Below, we start by summarizing our notion of "ad hoc freindliness," which (as we see later) is indeed satisfied by the above mentioned standard MIFE for inner product by [3, 2]. Our notion of ad hoc friendliness may be summarized as follows:

1. *Decentralized Setup.* The $\mathsf{MIFE.Setup}$ algorithm of the MIFE is decentralized in the sense that:
   a. The encryption keys EK_i for $i \in [n]$ corresponding to party i may be generated *independently* of the encryption keys of the remaining parties $[n] \setminus i$.
   b. The master secret key MSK can be decomposed into n components $\{\mathsf{MSK}_i\}_{i \in [n]}$. The partial MSK_i corresponding to party i may be generated locally by party i, without any interaction or shared state with the remaining parties.
2. *Local Encryption.* The encryption algorithm only takes its encryption key and message as input and does not depend on the number of parties or their public parameters.
3. *Piecewise Master Secret Key.* The master secret in standard MIFE $\mathsf{MSK} = \{\mathsf{MSK}_1, \ldots, \mathsf{MSK}_n\}$, if restricted to some subset $S \subseteq [n]$ with $|S| = \ell$, has the same distribution as a master secret generated for functions of arity ℓ.

We show that a standard MIFE with the above properties can be upgraded to ad hoc MIFE described above in a more direct manner than our generic transformation from standard MIFE to ad hoc MIFE. To see this, recall that one of the key challenges in ad hoc MIFE is that the encryptor must encrypt her messages without knowing the encryption key for the underlying standard MIFE. This is because the members or size of the group that will participate in the computation are chosen dynamically later.

To handle this, we used single input FE to encrypt messages and the MPC protocol for functionality $\mathsf{GenKeys}$ to sample an MIFE scheme and then translate the FE ciphertexts to MIFE ciphertexts. In the current setting however, due to properties (1) and (2) above, the encryption key of each party can be generated locally and each party can directly perform MIFE encryption locally. Since the re-encryption functionality involves computing a PRF and computing an MIFE encryption, the savings accrued by skipping this step are significant.

We will still require MPC to compute the MIFE function key, but no longer need the MPC functionality to sample the master secret key, so for simple functionalities this protocol may be much leaner than our general MPC protocol. For instance, in the case of inner products, we show that the required MPC protocol only needs to support inner product computations.

While the structural requirements described above may seem very strong, as mentioned earlier, we show that these requirements are enjoyed by the MIFE for inner products recently constructed by Abdalla et al. [2] and can be used to instantiate our compiler providing a very simple and efficient ad hoc MIFE for inner products. Please see Section 5 for details.

1.3.2.1 Instantiating MPC

As discussed for the case of our generic construction, we use two-round MPC protocols to obtain ad hoc MIFE for inner products. Specifically, since function-rerunnable two-round MPC in the common reference string (CRS) model can be constructed [28, 57, 24, 59] from learning-with-errors (LWE), we immediately get ad hoc MIFE from LWE. This result can be upgraded to the malicious setting at the additional cost of NIZKs. On the other hand, construction of two-round MPC from two-round oblivious transfer [15, 42, 39], yields *bounded* ad hoc MIFE for inner products under DDH, LWE or DCR, albeit with the requirement that the sources maintain state across key issues. One nice feature of these schemes is that they work without the need for a CRS in the semi-honest setting and upgrade to the malicious setting can be made just using CRS. Please see Section 5 for details.

Our results are highlighted in Table 1.

1.3.3 Related Work

In this section, we discuss the prior work in this area. Here, we focus on two related primitives, decentralized multi-client FE and and non-interactive MPC. Further related work on FE, MIFE, MPC and multi-authority FE can be found in Appendix A.

1.3.3.1 Decentralized Multi-Client Functional Encryption

Very recently, Chotard et al [27] proposed the notion of decentralized multi-client functional encryption (D-MCFE). While the motivation for the two works is similar in removing the common key authority, our notion of adhoc MIFE is significantly more general in that:

1. MCFE itself is more restricted than MIFE, since only CTs with the *same labels* can be combined. In MIFE there is no such restriction. MIFE for circuits captures MCFE for circuits (by checking for equal labels within the MIFE functionality) but not vice versa.
2. Crucially, the setup algorithm in D-MCFE is a protocol that is run between multiple senders, requiring interaction, whereas our setup algorithm is run independently by each source and is thus *non-interactive*. Note that developing a non-interactive solution is one of the main motivations of this work.
3. The work of Chotard et al [27] only provides a construction for inner products. We provide a general construction as well as one for inner products. Since our model is stronger, our inner product construction is significantly more involved than theirs.
4. Decentralized MCFE lacks the dynamic aspect, which is one of the main contributions of our work. We permit the function arity and participating parties to be chosen dynamically – a feature no other construction supports (to the best of our knowledge).

Non-Interactive MPC

Another related notion is that of *non-interactive MPC* (NI-MPC), where a group of asynchronous parties may evaluate a function over their inputs by sending a single message to an evaluator who computes the output [51]. While they appear superficially similar, we note that the model of ad hoc MIFE is fundamentally different from NI-MPC since, unlike NI-MPC, it separates inputs and functions, i.e. provides ciphertexts and function keys which allows *reusing* an input/ciphertext with many different functions, and a function with many different inputs. On the other hand, NI-MPC does not support function reusability at all, and only a very restricted version of input re-usability, namely where only ciphertexts in the same "session" may be combined. The function arity in NI-MPC is also fixed, unlike ad hoc MIFE.

2 Preliminaries

Due to space constraints, the preliminaries can be found in Appendix B.

3 *Ad hoc* Multi-Input Functional Encryption

We are now ready to define our new notion of *ad hoc* multi-input functional encryption (MIFE). For simplicity, we define ad hoc MIFE in the private-key setting only. We leave the study of ad hoc MIFE in the public-key setting for future work.

3.1 Syntax and Correctness

An *ad hoc multi-input functional encryption scheme* aMIFE for a message space $\{\mathcal{M}_\kappa\}_{\kappa\in\mathbb{N}}$ and a functionality $\{\mathcal{F}_\kappa\}_{\kappa\in\mathbb{N}}$, where for each $\kappa \in \mathbb{N}$, each $f \in \mathcal{F}_\kappa$ is a (description of a) function on $(\mathcal{M}_\kappa)^\ell$ for some ℓ (which may depend on f), is given by a set of algorithms with the following syntax:

- $\mathsf{aMIFE.Setup}(1^\kappa)$: A PPT algorithm taking the security parameter κ, and outputting the master secret key MSK and the public parameters PP.
- $\mathsf{aMIFE.KeyGen}(i, \mathsf{MSK}_i, (\mathsf{PP}_1, \cdots, \mathsf{PP}_\ell), f)$: A PT algorithm taking an index $i \in [\ell]$, a master secret key MSK_i corresponding to PP_i, a set of public parameters $\mathsf{PP}_1, \cdots, \mathsf{PP}_\ell$, a function $f \in \mathcal{F}_\kappa$ of arity ℓ, and outputting a corresponding partial decryption key $\mathsf{PDK}_{i,f}$.
- $\mathsf{aMIFE.Enc}(\mathsf{MSK}, \mathbf{x})$: A PPT algorithm taking a master secret key MSK and a message $\mathbf{x} \in \mathcal{M}_\kappa$, and outputting a ciphertext c.
- $\mathsf{aMIFE.Dec}((\mathsf{PDK}_{1,f}, \dots, \mathsf{PDK}_{\ell,f}), (c_1, \dots, c_\ell))$: A PT algorithm taking partial decryption keys $(\mathsf{PDK}_{1,f}, \dots, \mathsf{PDK}_{\ell,f})$ and ciphertexts $(c_1, \dots, c_\ell)$, and outputting a string y.

▶ **Definition 1** (Correctness). *We say that* aMIFE *is* correct *if for all* $\kappa \in \mathbb{N}$ *and* $\ell = \mathit{poly}(\kappa)$*, all* $\mathbf{x}_1 \dots \mathbf{x}_\ell \in \mathcal{M}_\kappa$ *and all* $f \in \mathcal{F}_\kappa$ *of arity* ℓ

$$\Pr\left[y = f(\mathbf{x}_1, \cdots, \mathbf{x}_\ell) \left| \begin{array}{l} (\mathsf{PP}_i, \mathsf{MSK}_i) \leftarrow\!\!\$\ \mathsf{aMIFE.Setup}(1^\kappa) \quad (\forall i \in [\ell]) \\ c_i \leftarrow\!\!\$\ \mathsf{aMIFE.Enc}(\mathsf{MSK}_i, \mathbf{x}_i) \\ \mathsf{PDK}_{i,f} \leftarrow\!\!\$\ \mathsf{aMIFE.KeyGen}(i, \mathsf{MSK}_i, (\mathsf{PP}_i)_{i\in[\ell]}, f) \\ y \leftarrow \mathsf{aMIFE.Dec}((\mathsf{PDK}_{i,f})_{i\in[\ell]}, (c_i)_{i\in[\ell]}) \end{array} \right. \right] = 1\,.$$

▶ Remark 2. We highlight two ways that ad hoc MIFE differs from standard MIFE [45]. First, the $\mathsf{aMIFE.Setup}$ algorithm is run per user and does not output all of the $\mathsf{MSK}_1, \dots \mathsf{MSK}_n$ at once. Second, the total number of users n and the function arity ℓ are not fixed and input to the $\mathsf{aMIFE.Setup}$ algorithm. We also note that for simplicity in our formulation of ad hoc MIFE the public parameters of the parties input to the key generation algorithm are ordered.

▶ Remark 3. We also consider two relaxations of ad hoc MIFE:

- We can allow an additional algorithm CRSGen taking 1^κ and outputting a common reference string CRS that is input to the remaining algorithms. We refer to this as ad hoc MIFE *in the CRS model.* The CRS model is weaker than having a key generation authority who can decrypt all the data.
- We can allow the total number of users 1^n to be input to the setup algorithm. We refer to this as *bounded* ad hoc MIFE. We can additionally require $n = \ell$, which we refer to as *static* (vs. dynamic) ad hoc MIFE. In particular, static ad hoc MIFE recovers a variant of MIFE that is similar to the standard one but still eliminates the trusted key generation authority.

3.2 Indistinguishability-Based Security

We first present an indistinguishability-based security notion. We note that the fact that the public parameters of the parties input to the key generation algorithm are ordered allows us to work with a somewhat simpler definition than the corresponding one in [45] for the standard MIFE case.

For an ad hoc MIFE scheme aMIFE as above and adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$, consider the experiment in Figure 1.

For an ad hoc MIFE scheme aMIFE as above and adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$, consider the experiment in Figure 1.

```
Experiment $\mathbf{IND}_{\mathcal{A}}^{\mathsf{aMIFE}}(1^\kappa)$
    $(1^n, I, (\mathsf{PP}_i)_{i \in I}, st) \leftarrow\$ \mathcal{A}_0(1^\kappa)$
    $b \leftarrow\$ \{0,1\}$
    For all $i \notin I$ do:
        $(\mathsf{MSK}_i, \mathsf{PP}_i) \leftarrow\$ \mathsf{aMIFE.Setup}(1^\kappa)$
    $st \leftarrow\$ \mathcal{A}_1^{\mathcal{O}_{\mathsf{enc}}(\cdot,\cdot,\cdot), \mathcal{O}_{\mathsf{kg}}(\cdot,\cdot,\cdot)}(st)$
    $b' \leftarrow\$ \mathcal{A}_2^{\mathcal{O}_{\mathsf{enc}}(\cdot,\cdot,\cdot), \mathcal{O}_{\mathsf{kg}}(\cdot,\cdot,\cdot)}((\mathsf{PP}_i)_{i \notin I}, st)$
    Return $(b = b')$
```

```
Oracle $\mathcal{O}_{\mathsf{enc}}(i, \mathbf{x}_0, \mathbf{x}_1)$
    If $\mathbf{x}_0, \mathbf{x}_1 \in \mathcal{M}_\kappa$ and $|\mathbf{x}_0| = |\mathbf{x}_1|$
        $c_b \leftarrow\$ \mathsf{aMIFE.Enc}(\mathsf{MSK}_i, \mathbf{x}_b)$
        Return $c_b$
    Else Return $\perp$

Oracle $\mathcal{O}_{\mathsf{kg}}(i, (i_1, \ldots, i_\ell), f)$
    If $i \in \{i_1, \cdots, i_\ell\}$
    and $f \in \mathcal{F}_\kappa$ of arity $\ell$
        $\mathsf{PDK}_{i,f} \leftarrow$
            $\mathsf{aMIFE.KeyGen}(i, \mathsf{MSK}_i, (\mathsf{PP}_{i_j}), f)$
        Return $\mathsf{PDK}_{i,f}$
    Else return $\perp$
```

Figure 1 Experiment for IND-security of ad hoc MIFE.

We say that $f \in \mathcal{F}_\kappa$ is *queried* if for every user associated with its input wires, either the user is corrupt, or has submitted it's partial decryption key to the adversary. Formally, for every $k \in \ell$ where ℓ is the arity of f either $i_k \in I$, *i.e.* i_k is corrupted or there is a key-generation query $(i_k, (\mathsf{PP}_{i_1}, \cdots, \mathsf{PP}_{i_\ell}), f)$.

We call $\mathcal{A}$ *legitimate* if for all $\kappa \in \mathbb{N}$, in all transcripts $\mathbf{IND}_{\mathcal{A}}^{\mathsf{aMIFE}}(1^\kappa)$ it holds that for every queried $f \in \mathcal{F}_\kappa$, there does not exist two sequences $(y_{i_1,0}, \cdots, y_{i_\ell,0})$ and $(y_{i_1,1}, \cdots, y_{i_\ell,1})$ such that

$$f(y_{i_1,0}, \cdots, y_{i_\ell,0}) \neq f(y_{i_1,1}, \cdots, y_{i_\ell,1})$$

and for every $j \in \{i_1, \cdots, i_\ell\}$

- $j \in I$, *i.e.* j is corrupted (so there is no restriction on $y_{j,0}, y_{j,1}$ above), or
- There is an encryption query $(j, \mathbf{x}_0, \mathbf{x}_1)$ such that $y_{j,0} = \mathbf{x}_0$ and $y_{j,1} = \mathbf{x}_1$.

We assume adversaries are legitimate unless otherwise stated. We call $\mathcal{A}$ *passive* if $I = \emptyset$. We call $\mathcal{A}$ *selective* if $\mathcal{A}_2$ makes no queries. We say that aMIFE is xxx-IND-secure if for any adversary $\mathcal{A}$ of type xxx

$$|\Pr\left[\mathbf{IND}_{\mathcal{A}}^{\mathsf{aMIFE}}(\cdot) \text{ outputs } 1\right] - 1/2| = \mathsf{negl}(\cdot) \; .$$

We provide the definition of simulation based security in Appendix C.

4 Ad Hoc MIFE from MIFE + Two-Round MPC

We show how to construct of ad hoc MIFE for any polynomial sized circuit from standard MIFE for the same functionality and a two-round MPC protocol.

Building Blocks

Our scheme will be using the following building blocks:

- A MIFE scheme

  $$\mathsf{MIFE} = (\mathsf{MIFE.Setup}, \mathsf{MIFE.KeyGen}, \mathsf{MIFE.Enc}, \mathsf{MIFE.Dec})$$

  for some message-space $\{\mathcal{M}_\kappa\}_{\kappa\in\mathbb{N}}$ and functionality $\{\mathcal{F}_\kappa\}_{\kappa\in\mathbb{N}}$. For simplicity, we assume $\mathsf{MIFE.KeyGen}$ is deterministic; note that this is without loss of generality since it can be made so by using a PRF.
- A two-round two-round MPC protocol

  $$\mathsf{MPC} = (\mathsf{MPC.RunRoundOne}, \mathsf{MPC.RoundRoundTwo}, \mathsf{MPC.ComputeResult})$$

  for programs of the form $\mathsf{GenKeys}_f$ in Figure 2 for $f \in \mathcal{F}_\kappa$. We assume MPC is function-rerunnable, unbounded and without setup (we discuss the other cases below).
- A PRF F with keyspace $\{\mathcal{K}_\kappa\}_{\kappa\in\mathbb{N}}$, and a punctured PRF puncF with keyspace $\{\mathcal{K}^{\mathsf{punc}}_\kappa\}_{\kappa\in\mathbb{N}}$, both with domain $\{0,1\}^*$. (We leave the ranges implicit for readability, taking the output to be sufficiently long.)
- A private-key single input functional encryption scheme

  $$\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.KeyGen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$$

4.1 Construction

Below we provide our construction for adhoc MIFE for general circuits. Note that setup, encryption and key generation are done independently and in parallel by all the parties in the system.

$\mathsf{aMIFE.Setup}(1^\kappa)$: Upon input the security parameter, do the following:

1. Sample the seed of PRF $K \leftarrow_{\$} \mathcal{K}_\kappa$ and the seed of a puncturable PRF $K^{\mathsf{punc}} \leftarrow_{\$} \mathcal{K}^{\mathsf{punc}}_\kappa$. Puncturing will only be required in the proof.
2. Invoke the single input FE scheme, $(\mathsf{PP}_{\mathsf{FE}}, \mathsf{MSK}_{\mathsf{FE}}) \leftarrow_{\$} \mathsf{FE.Setup}(1^\kappa)$.
3. Invoke the first round of the MPC protocol

   $$(\rho^{(1)}, \mathfrak{s}) \leftarrow_{\$} \mathsf{MPC.RunRoundOne}(1^\kappa, (K, \mathsf{MSK}_{\mathsf{FE}}))$$

   Note that the function is specified later.
4. Return $(\mathsf{PP} = \rho^{(1)}, \mathsf{MSK} = (K, K^{\mathsf{punc}}, \mathsf{MSK}_{\mathsf{FE}}, \mathfrak{s}))$.

$\mathsf{aMIFE.KeyGen}((\mathsf{PP}_i)_{i\in[\ell]}, f, \mathsf{MSK})$: Upon input the public parameters of the ℓ parties that are chosen to participate in the computation, and the master secret key, do the following:

1. Parse the public parameters of each party as the first message in an MPC protocol, i.e. $\rho_i^{(1)} \leftarrow \mathsf{PP}_i \quad \forall i \in [\ell]$.
2. Parse the master secret key as $(K, K^{\mathsf{punc}}, \mathsf{MSK}_{\mathsf{FE}}, \mathfrak{s}) \leftarrow \mathsf{MSK}$
3. Run round two of the MPC protocol using round 1 messages as input, for the functionality $\mathsf{GenKeys}$ described in Figure 2:
$$\rho^{(2)} \leftarrow\!\!\$\ \mathsf{MPC.RunRoundTwo}(\mathfrak{s}, \mathsf{GenKeys}_{(\mathsf{PP}_i)_{i\in[\ell]}, f}, (\rho_i^{(1)})_{i\in[\ell]})$$
4. Compute the mask $s \leftarrow \mathsf{PRF.Eval}(K, 0 \parallel (\mathsf{PP}_i)_{i\in[\ell]} \parallel f)$
5. Return $(\rho^{(2)}, s)$.

$\mathsf{aMIFE.Enc}(\mathsf{MSK}, \mathbf{x})$: Upon input the master secret key and the message $\mathbf{x}$, do the following:

1. Parse the master secret key as $(K, K^{\mathsf{punc}}, \mathsf{MSK}_{\mathsf{FE}}, \mathfrak{s}) \leftarrow \mathsf{MSK}$.
2. Sample the tag $T \leftarrow\!\!\$\ \{0,1\}^\kappa$.
3. Initialize the data structure Trap defined in Figure 10 by setting $\mathsf{mode\text{-}real} = 1$ and all other fields as $\perp$. This indicates that we are in the real system. The remaining fields are only relevant in the proof.
4. Compute the ciphertext $c \leftarrow\!\!\$\ \mathsf{FE.Enc}(\mathsf{MSK}_{\mathsf{FE}}, (\mathbf{x}, T, K^{\mathsf{punc}}, \mathsf{Trap}))$.
5. Return c.

$\mathsf{aMIFE.Dec}((\mathsf{PDK}_{i,f})_{i\in[\ell]}, (c_i)_{i\in[\ell]})$: Upon input the partial decryption keys from all relevant parties, as well as ciphertexts from all relevant parties, do the following:

1. Parse $(\rho_i^{(2)}, s_i) \leftarrow \mathsf{PDK}_{i,f} \quad \forall i \in [\ell]$
2. Compute the output of the MPC protocol as $Z \leftarrow \mathsf{MPC.ComputeResult}((\rho_i^{(2)})_{i\in[\ell]})$
3. Unmask the output using partial shares provided by all parties. In more detail, compute $S \leftarrow \bigoplus_{i\in[\ell]} s_i$; $Z \leftarrow Z \oplus S$.
4. Parse the output of the MPC computation as $(\mathsf{SK}_f, \mathsf{SK}_{\mathsf{FE}_1}, \ldots, \mathsf{SK}_{\mathsf{FE}_\ell}) \leftarrow Z$.
5. Perform decryption of the single input FE scheme to obtain MIFE ciphertexts $\psi_i \leftarrow \mathsf{FE.Dec}(\mathsf{SK}_{\mathsf{FE}_i}, c_i) \quad \forall i \in [\ell]$.
6. Perform decryption of the MIFE scheme to obtain the output $y \leftarrow \mathsf{MIFE.Dec}(\mathsf{SK}_f, \psi_1, \ldots, \psi_\ell)$.
7. Return y.

▶ Remark 4. If we use a bounded 2-round MPC protocol then we will obtain a bounded ad hoc MIFE scheme where the setup algorithm also takes 1^n which is passed to $\mathsf{MPC.RunRoundOne}$. If MPC has a setup algorithm (outputting a CRS) then so does the resulting ad hoc MIFE scheme.

Correctness

Correctness follows from the correctness of MPC, MIFE and FE. In more detail, we have:

Step 1: MPC. By correctness of MPC, we have that the decryptor recovers the output $S \oplus (\mathsf{SK}_f, \mathsf{SK}_{\mathsf{FE}_1}, \ldots, \mathsf{SK}_{\mathsf{FE}_\ell})$. Next, if each party $i \in [\ell]$ provides a partial decryption key for f, then this contains partial mask s_i as part of the output of $\mathsf{aMIFE.KeyGen}$. Using these, the decryptor can compute $S \leftarrow \bigoplus_{i\in[\ell]} s_i$ and recover $(\mathsf{SK}_f, \mathsf{SK}_{\mathsf{FE}_1}, \ldots, \mathsf{SK}_{\mathsf{FE}_\ell})$.

```
Function $\mathsf{GenKeys}_{(\mathsf{PP}_i)_{i\in[\ell]},f}((K_1, \mathsf{MSK}_{\mathsf{FE}_1}), \ldots, (K_\ell, \mathsf{MSK}_{\mathsf{FE}_\ell}))$
1. For $i \in [\ell]$, compute randomness to be used for algorithms below:
       $r_i \leftarrow \mathsf{PRF.Eval}(K_i, 1 \parallel (\mathsf{PP}_i)_{i\in[\ell]} \parallel f)$,    $r'_i \leftarrow \mathsf{PRF.Eval}(K_i, 2 \parallel (\mathsf{PP}_i)_{i\in[\ell]} \parallel f)$
2. Run the MIFE setup algorithm for the desired arity as:
       $((\mathsf{EK}_1, \ldots, \mathsf{EK}_\ell), \mathsf{MSK}) \leftarrow \mathsf{MIFE.Setup}(1^\kappa; r_1 \oplus \cdots \oplus r_\ell)$
3. For $i \in [\ell]$, generate the single input FE function key:
       $\mathsf{SK}_{\mathsf{FE}_i} \leftarrow \mathsf{FE.KeyGen}(\mathsf{MSK}_{\mathsf{FE}_i}, \mathsf{ReEnc}_{\mathsf{EK}_i,\perp}; r'_i)$
4. Compute the MIFE secret key as $\mathsf{SK}_f \leftarrow \mathsf{MIFE.KeyGen}(\mathsf{MSK}, f)$
5. Sample the mask $s_i$ for each partial key and compute the mask $S$ as:
       $\forall i \in [\ell],\ \ s_i \leftarrow \mathsf{PRF.Eval}(K_i, 0 \parallel (\mathsf{PP}_i)_{i\in[\ell]} \parallel f)$,    $S \leftarrow \bigoplus_{i\in[\ell]} s_i$
6. Return the masked output $S \oplus (\mathsf{SK}_f, \mathsf{SK}_{\mathsf{FE}_1}, \ldots, \mathsf{SK}_{\mathsf{FE}_\ell})$
```

Figure 2 Functionality computed by the MPC protocol to generate single and multi input FE keys.

```
Function $\mathsf{ReEnc}_{\mathsf{EK},\mathsf{ctr}}(\mathbf{x}, T, K, \mathsf{Trap})$
1. Let $r \leftarrow \mathsf{PRF.Eval}(K, 0 \parallel \mathsf{EK} \parallel T)$.
2. If $\mathsf{mode} = R$ then compute $\mathsf{MIFE.Enc}(\mathsf{EK}, \mathbf{x}; r)$ and return it.
3. If $\mathsf{mode} = T$ and $\mathsf{ctr} < \mathsf{Trap.index}$ then compute $\mathsf{MIFE.Enc}(\mathsf{EK}, \mathsf{Trap.val}_1; r)$ and return it.
4. If $\mathsf{mode} = T$ and $\mathsf{ctr} > \mathsf{Trap.index}$ then compute $\mathsf{MIFE.Enc}(\mathsf{EK}, \mathsf{Trap.val}_0; r)$ and return it.
5. If $\mathsf{mode} = T$ and $\mathsf{ctr} = \mathsf{Trap.index}$ then output $\mathsf{Trap.CT}$.
```

Figure 3 Functionality for translating the ciphertext from FE to MIFE using dynamically generated encryption keys.

Step 2: FE. Next, by correctness of FE, we have that if

$$\psi_i = \mathsf{FE.Dec}(\mathsf{SK}_{\mathsf{FE}_i}, c_i) \quad \forall i \in [\ell]$$

Then, ψ_i are the MIFE ciphertexts computed as $\mathsf{MIFE.Enc}(\mathsf{EK}_i, \mathbf{x}_i; r_i)$.

Step 3: MIFE. Finally, by correctness of MIFE, we have that

$$f(\mathbf{x}_1, \ldots, \mathbf{x}_\ell) = \mathsf{MIFE.Dec}(\mathsf{SK}_f, \psi_1, \ldots, \psi_\ell)$$

4.2 Security Proof

In this section, we argue that the scheme described above is secure. In more detail:

▶ **Theorem 5.** *If* MIFE *is a IND-secure MIFE scheme and* MPC *is a SIM-secure 2-round MPC protocol, then our construction is* sel-*IND-secure.*

Due to space constraints, the proof can be found in Appendix D.

5 Ad Hoc Friendly MIFE and its Application to Inner Products

In this section, we describe a paradigm for constructing ad-hoc MIFE schemes from MIFE schemes that are "ad hoc friendly" and a (hopefully simple) two-round MPC protocol. This paradigm significantly simplifies our general construction. We then show that the standard MIFE scheme for inner products [3, 2], which may be based on DDH, LWE or DCR, is ad hoc friendly and the corresponding two-round MPC protocol is only required to compute inner-products, thus obtaining an efficient ad hoc MIFE scheme for inner products.

Ad Hoc Friendliness

In more detail, we define a notion of "ad hoc friendly" standard MIFE which satisfies the following properties:

- **Decentralized Setup.** The $\mathsf{MIFE.Setup}$ algorithm of the MIFE is decentralized in the sense that:
  1. The encryption keys EK_i for $i \in [n]$ corresponding to party i may be generated *independently* of the encryption keys of the remaining parties $[n] \setminus i$. In more detail, the algorithm $(\mathsf{EK}_1, \dots, \mathsf{EK}_n) \leftarrow \mathsf{MIFE.Setup}(1^\kappa)$ may be decomposed into n invocations $\mathsf{EK}_i \leftarrow \mathsf{MIFE.SetupLocal}(1^\kappa)$ for $i \in [n]$, which can be run locally by each party.
  2. The master secret key MSK can be decomposed into n components $\{\mathsf{MSK}_i\}_{i\in[n]}$. The partial MSK_i corresponding to party i may be generated locally by party i, without any interaction or shared state with the remaining parties.

- **Local Encryption.** The $\mathsf{MIFE.Enc}$ algorithm of the MIFE is "local" in that it does not take as input the total number of parties or the public parameters of other parties. In more detail, $\mathsf{MIFE.Enc}$ algorithm only takes as input its encryption key EK_i and its input $\mathbf{x}_i$, and nothing else.

- **Piecewise Master Secret Key.** In standard MIFE schemes, the function is assumed to have fixed arity n. However, in ad hoc MIFE, we allow the function to have arity $\ell < n$. To support this, we require that the master secret in standard MIFE $\mathsf{MSK} = \{\mathsf{MSK}_1, \dots, \mathsf{MSK}_n\}$, if restricted to some subset $S \subseteq [n]$ with $|S| = \ell$, has the same distribution as a master secret generated for functions of arity ℓ.

  Formally, let $\mathsf{MSK} = \{\mathsf{MSK}_1, \dots, \mathsf{MSK}_n\} \leftarrow \mathsf{FE.Setup}(1^\kappa, 1^n)$ and $\mathsf{MSK}' = \{\mathsf{MSK}'_1, \dots, \mathsf{MSK}'_\ell\} \leftarrow \mathsf{FE.Setup}(1^\kappa, 1^\ell)$. Then, we require that MSK restricted to subset S, namely $(\mathsf{MSK}_{S[1]}, \dots, \mathsf{MSK}_{S[\ell]})$ has the same distribution as MSK'.

Since the intuition was discussed in Section 1, we proceed to our construction of ad hoc MIFE for inner products.

5.1 Ad Hoc MIFE for Inner Products

Inner-product functionality

We recall the *multi-input inner-product functionality* over $\mathbb{Z}_p$ for a prime p, adapted from Abdalla et al. [2, Section 2.3]. For $m, n \in \mathbb{N}$, this is the functionality

$$\mathcal{IP}^m_{p,n} = \{\mathsf{ip}_{\mathbf{y}_1,\dots,\mathbf{y}_n} \colon (\mathbb{Z}_p^m)^n \to \mathbb{Z}_p\}$$

defined by

$$\mathsf{ip}_{\mathbf{y}_1,\ldots,\mathbf{y}_n}(\mathbf{x}_1,\ldots,\mathbf{x}_n) \;=\; \sum_{i=1}^{n}\langle \mathbf{x}_i, \mathbf{y}_i\rangle \bmod p\ .$$

We omit parameters p, m, n when they are arbitrary or clear from context.

5.2 Building Blocks

In the context of ad hoc MIFE for inner products, we want to evaluate a functionality given by a sequence of vectors $\mathbf{y} = (\mathbf{y_1}, \ldots, \mathbf{y_n})$ where $\mathbf{y_i} \in \mathbb{Z}_q^m$. Evaluating the function on inputs $\{\mathbf{x_i}\}_{i\in[n]}$ where $\mathbf{x_i} \in \mathbb{Z}_q^m$ reveals $\sum_{i\in[n]}\langle \mathbf{x_i}, \mathbf{y_i}\rangle$ and nothing more. In particular the evaluator should not be able to learn the partial sums $\langle \mathbf{x_i}, \mathbf{y_i}\rangle$.

Our scheme will be using the following building blocks:

- A 2-round *function-rerunnable MPC for a functionality* $\mathsf{GenKey\text{-}ip}$*, which must be support inner product computation.*
- *A standard MIFE with for the inner product functionality, denoted by* $\mathsf{MIFE_{ip}}$*, satisfying the aforementioned ad hoc friendly properties.*

The MIFE scheme(s) of Abdalla et al. [2]:

Abdalla et al. [2] provide two multi-input encryption schemes for inner products, one for computing inner products over some finite ring $\mathbb{Z}_L$, and the second for computing bounded-norm inner products over the integers. Both schemes rely on:

1. An information theoretic scheme for inner products where only one ciphertext query is supported.
2. A single input functional encryption scheme FE for inner products which is applied on top of the above one time scheme.

Unrolling the above two components, the final MIFE scheme(s) of [2] have algorithms of the form described below.

Below, we unroll the above two components to establish that the schemes of [2] satisfy ad hoc friendliness.

1. **Decentralized Setup.** The encryption keys EK_i corresponding to party i may be generated *independently* of the encryption keys of the remaining parties $[n] \setminus i$. In more detail, the setup algorithm is defined as:

   $\mathsf{MIFE.Setup}(1^\kappa, n)$: Do the following:
   - For $i \in [n]$, sample $\mathbf{u}_i \leftarrow \mathbb{Z}_L^m$.
   - For $i \in [n]$, sample $(\mathsf{FE.PK}_i, \mathsf{FE.MSK}_i) \leftarrow \mathsf{FE.Setup}(1^\kappa, 1^m)$.
   - Output $\mathsf{PP}_i = \mathsf{FE.PK}_i$ and $\mathsf{EK}_i = (\mathsf{FE.MSK}_i,\ \mathbf{u}_i)$ for $i \in [n]$.

   Then, we may define:

   a. $\mathsf{MIFE.SetupLocal}(1^\kappa, n)$: Do the following:
      - Sample $\mathbf{u} \leftarrow \mathbb{Z}_L^m$.
      - Sample $(\mathsf{FE.PK}, \mathsf{FE.MSK}) \leftarrow \mathsf{FE.Setup}(1^\kappa, 1^m)$.
      - Output $\mathsf{PP} = \mathsf{FE.PP}$ and $\mathsf{EK} = (\mathsf{FE.MSK},\ \mathbf{u})$.

To compute the set of n encryption keys, the algorithm $\mathsf{MIFE.SetupLocal}(1^\kappa, n)$ is invoked n times. Additionally, in [2], the master secret key can be decomposed into n components by setting:

$$\mathsf{MSK}_i = \mathsf{EK}_i = (\mathsf{FE.MSK}_i, \mathbf{u}_i)\ \forall\ \ i \in [n]$$

2. **Local Encryption.** The encryption algorithm only takes its encryption key and message as input and does not depend on the number of parties or their public parameters. In more detail, the encryption algorithm is defined as:

   $\mathsf{MIFE.Enc}(\mathsf{EK}_i, \mathbf{x}_i)$: Do the following:
   - Parse $\mathsf{EK}_i = (\mathsf{FE.MSK}_i,\ \mathbf{u}_i)$.
   - Compute $\mathbf{y}_i = \mathbf{x}_i + \mathbf{u}_i \mod L$.
   - Compute $\mathbf{c}_i = \mathsf{FE.Enc}(\mathsf{FE.MSK}_i, \mathbf{y}_i)$.
   - Output $(\mathbf{y}_i, \mathbf{c}_i)$.

   Thus, the ciphertext encoding party i's input may be computed independently by party i.

3. **Piecewise Master Secret Key.** For the inner product functionality, if $\mathsf{MSK} = (\mathsf{MSK}_1, \ldots, \mathsf{MSK}_n)$ is the master secret key for function vector $\mathbf{y} = (\mathbf{y}_1 \| ... \| \mathbf{y}_n)$ then for any $S \subseteq [n]$, we have the corresponding master key $\mathsf{MSK}' = (\mathsf{MSK}_{S[1]}, \ldots, \mathsf{MSK}_{S[\ell]})$ is a well formed master secret key for the vector $\mathbf{y}' = (\mathbf{y}_{S[1]} \| \ldots \| \mathbf{y}_{S[\ell]})$. In more detail, the key generation algorithm is defined as:

   $\mathsf{MIFE.KeyGen}(\mathsf{MSK}, \mathbf{y})$: Do the following:
   - Output $\mathsf{DK}_{\mathbf{y}} \leftarrow \big(\ \{\mathsf{FE.KeyGen}(\mathsf{MSK}_i, \mathbf{y}_i)\}_{i \in [n]},\ \sum\limits_{i \in [n]} \langle \mathbf{u}_i,\ \mathbf{y}_i \rangle\ \big)$.

   It is easy to see that the function key for $\mathbf{y}'$ can be obtained from the above by simply setting $\mathbf{y}_i = 0$ for $i \notin S$.

5.3 Our Construction

We are now ready to present the construction. Note that for ease of presentation, we describe the scheme for all n users but we remark that it works for any subset of $\ell \leq n$ users.

$\mathsf{aMIFE.Setup}(1^\kappa, 1^m)$: Upon input the security parameter and the dimension of the input vector for each party, do the following:

1. Run the partial MIFE setup algorithm to obtain the public parameters and encryption key: $(\mathsf{MIFE.PP}, \mathsf{MIFE.EK}) \leftarrow\!\!\$\ \mathsf{MIFE.SetupLocal}(1^\kappa, 1^m)$
2. Invoke the first round of the MPC protocol with the encryption key as input:

$$(\rho^{(1)}, \mathfrak{s}) \leftarrow\!\!\$\ \mathsf{MPC.RunRoundOne}(1^\kappa, \mathsf{EK})$$

3. Return $\mathsf{PP} := (\mathsf{MIFE.PP}, \rho^{(1)}), \mathsf{MSK} := (\mathsf{MIFE.EK}, \mathfrak{s})$

$\mathsf{aMIFE.Enc}(\mathsf{EK}, \mathrm{x})$: Upon input the encryption key and the input, compute $\mathsf{MIFE.enc}(\mathsf{EK}, \mathbf{x})$ and output it.

$\mathsf{aMIFE.KeyGen}((\mathsf{PP}_i)_{i \in [\ell]}, \mathrm{y}, \mathsf{MSK}_i)$: Upon input the public parameters of the ℓ parties, the function vector $\mathbf{y}$ and the master secret key MSK_i, do the following:

1. Parse $(\mathsf{MIFE.EK}, \mathfrak{s}) \leftarrow \mathsf{MSK}_i$ and $(\mathsf{MIFE.PP}_j, \rho_j^{(1)}) \leftarrow \mathsf{PP}_j \quad \forall j \in [\ell]$
2. Parse $(\mathbf{y_1}, \ldots, \mathbf{y}_\ell) \leftarrow \mathbf{y}$ where $\mathbf{y_j} \in \mathbb{Z}_q^m$ for $j \in [\ell]$.

3. Invoke round 2 of the MPC protocol $\mathsf{GenKey\text{-}ip}_{\mathbf{y}}$ as defined in Figure 4

$$\rho^{(2)} \leftarrow_{\$} \mathsf{MPC.RunRoundTwo}(\mathfrak{s}, \rho_1^{(1)}, \ldots, \rho_\ell^{(1)},)$$

4. Return $\mathsf{PDK} := \rho^{(2)}$.

$\mathsf{aMIFE.Dec}((\mathsf{PDK}_{i,f})_{i\in[\ell]}, (\mathbf{c}_i)_{i\in[\ell]})$: Upon input the partial decryption keys from all relevant parties, as well as ciphertexts from all relevant parties, do the following:

1. Compute the output of the MPC protocol as

$$\mathsf{MIFE.DK}_{\mathbf{y}} \leftarrow \mathsf{MPC.ComputeResult}((\rho_i^{(2)})_{i\in[\ell]})$$

2. Compute $\mathsf{MIFE.Dec}(\mathsf{MIFE.DK}_{\mathbf{y}}, \mathbf{y}, \mathbf{c}_1, \ldots, \mathbf{c}_\ell)$ and output it.

Function $\mathsf{GenKey\text{-}ip}_{\mathbf{y}}(\mathsf{EK}_1, \ldots, \mathsf{EK}_\ell)$

1. Let $\mathsf{MIFE.MSK} = (\mathsf{EK}_1, \ldots, \mathsf{EK}_\ell)$.
2. Compute $\mathsf{MIFE.KeyGen}(\mathsf{MIFE.MSK}, \mathbf{y})$ and output it.

Figure 4 Functionality for computing the MIFE function key.

Note that for the inner product functionality, the MIFE key generation algorithm is very simple and in some cases only involves computing inner products, please see [2] for details.

Correctness

Correctness follows from correctness of the MPC protocol and of the standard MIFE scheme. We have by correctness of the MPC protocol, that the output $\mathsf{MIFE.DK}_{\mathbf{y}} = \mathsf{MIFE.KeyGen}(\mathsf{MSK}, \mathbf{y})$ is produced correctly. Since the encryptors encrypted $\mathbf{c}_i = \mathsf{MIFE.Enc}(\mathsf{EK}_i, \mathbf{x}_i)$, it follows from the correctness of MIFE that $\mathsf{MIFE.Dec}(\mathsf{MIFE.DK}_{\mathbf{y}}, \mathbf{y}, \mathbf{c}_1, \ldots, \mathbf{c}_\ell)$ outputs $\sum\limits_{i\in[\ell]} \langle \mathbf{y}_i,\ \mathbf{x}_i \rangle$ as desired.

Security

Given the proof of security in Section 4, the proof of security of the present construction is straightforward, since the present construction is a (much) simplified instance of the general construction. Intuitively, the security of MPC ensures that the output $\mathsf{MIFE.DK}_{\mathbf{y}}$, which is computed using inputs $(\mathsf{MSK}_i, \mathbf{y}_i)_{i\in[\ell]}$ of ℓ disjoint parties, is indistinguishable from the output of a "global" MIFE key generation algorithm which takes the entire $(\mathsf{MSK}, \mathbf{y})$ as input. The encryption algorithm is exactly the same as that of the standard MIFE scheme, with the result that the decryptor sees exactly the same view as in the standard MIFE scheme. Please see Appendix E for details.

An issue with the MIFE scheme of Abdalla et al. [2] which we use above is that an adversary may exploit partial ciphertexts to learn unauthorized information [43]. Specifically, say there are two parties, and the first one provides ciphertexts, for vectors $\mathbf{x}_0$ and $\mathbf{x}_1$ (say). The second encryptor does not give any ciphertexts. Suppose the key generation algorithm in the standard MIFE scheme gives a key for $(\mathbf{y} \parallel \mathbf{0})$. Now, it could be that $\langle \mathbf{x}_0, \mathbf{y} \rangle \neq \langle \mathbf{x}_1, \mathbf{y} \rangle$ but this does not violate admissibility, because admissibility only checks for decryption with respect to complete ciphertexts whereas decryption with respect to partial ciphertexts is not defined. Ideally, the key for $(\mathbf{y} \parallel \mathbf{0})$ should not work to decrypt the partial ciphertexts of encryptor 1, but in the construction of Abdalla et al., it does so (if the corresponding sub-vector in the key is zero).

Note that in a MIFE scheme, the above situation only occurs if one party (party 2 say) *never* gives any ciphertext[5]. In our setting however, a party issues a partial decryption key only if it wishes its data to participate in some computation. Hence, we resolve the issue by requiring that a party only issue a partial decryption key if it has also issued at least one ciphertext.

Instantiating MPC

Since function-rerunnable two-round MPC in the common reference string (CRS) model can be constructed [28, 57, 24, 59] from learning-with-errors (LWE), we get ad hoc MIFE for inner products from LWE. This result can be upgraded to the malicious setting as the additional cost of NIZKs.

While function rerunnable two-round MPC in the CRS model can be constructed from bilinear maps [41] and even two-round oblivious transfer [15, 42, 39] or information theoretically [9, 36], these constructions are *not* function-rerunnable so do not suffice for multi-key ad hoc MIFE. However, if we restrict ourselves to the setting of bounded ad hoc MIFE, where a user issues only a bounded number of partial decryption keys and additionally maintains state across key issues, we may use the above MPC protocols (just via repetition). This yields such a bounded ad hoc MIFE under DDH, LWE or DCR for the both the semi-honest and malicious cases. One nice feature of the semi-honest construction is that it does not use a common random string and is in the plain model.

References

1 Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. Simple Functional Encryption Schemes for Inner Products. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 733–751. Springer, Heidelberg, March/April 2015. doi:10.1007/978-3-662-46447-2_33.

2 Michel Abdalla, Dario Catalano, Dario Fiore, Romain Gay, and Bogdan Ursu. Multi-Input Functional Encryption for Inner Products: Function-Hiding Realizations and Constructions Without Pairings. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 597–627. Springer, Heidelberg, August 2018. doi:10.1007/978-3-319-96884-1_20.

3 Michel Abdalla, Romain Gay, Mariana Raykova, and Hoeteck Wee. Multi-input Inner-Product Functional Encryption from Pairings. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 601–626. Springer, Heidelberg, April/May 2017. doi:10.1007/978-3-319-56620-7_21.

4 Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient Lattice (H)IBE in the Standard Model. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 553–572. Springer, Heidelberg, May/June 2010. doi:10.1007/978-3-642-13190-5_28.

5 Shweta Agrawal, David Mandell Freeman, and Vinod Vaikuntanathan. Functional Encryption for Inner Product Predicates from Learning with Errors. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 21–40. Springer, Heidelberg, December 2011. doi:10.1007/978-3-642-25385-0_2.

6 Shweta Agrawal, Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional Encryption: New Perspectives and Lower Bounds. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 500–518. Springer, Heidelberg, August 2013. doi:10.1007/978-3-642-40084-1_28.

[5] If party 2 gives even a single ciphertext (say for vector $\mathbf{z}_0$) then admissibility will force $\langle \mathbf{x}_0 \parallel \mathbf{z}_0, \mathbf{y} \parallel \mathbf{0} \rangle = \langle \mathbf{x}_1 \parallel \mathbf{z}_0, \mathbf{y} \parallel \mathbf{0} \rangle$ which implies that $\langle \mathbf{x}_0, \mathbf{y} \rangle = \langle \mathbf{x}_1, \mathbf{y} \rangle$.

7 Shweta Agrawal, Benoît Libert, and Damien Stehlé. Fully Secure Functional Encryption for Inner Products, from Standard Assumptions. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 333–362. Springer, Heidelberg, August 2016. doi:10.1007/978-3-662-53015-3_12.

8 Prabhanjan Ananth and Abhishek Jain. Indistinguishability Obfuscation from Compact Functional Encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 308–326. Springer, Heidelberg, August 2015. doi:10.1007/978-3-662-47989-6_15.

9 Benny Applebaum, Zvika Brakerski, and Rotem Tsabary. Perfect Secure Computation in Two Rounds. In *TCC 2018, Part I*, LNCS, pages 152–174. Springer, Heidelberg, March 2018. doi:10.1007/978-3-030-03807-6_6.

10 Saikrishna Badrinarayanan, Divya Gupta, Abhishek Jain, and Amit Sahai. Multi-input Functional Encryption for Unbounded Arity Functions. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 27–51. Springer, Heidelberg, November/December 2015. doi:10.1007/978-3-662-48797-6_2.

11 Carmen Elisabetta Zaira Baltico, Dario Catalano, Dario Fiore, and Romain Gay. Practical Functional Encryption for Quadratic Functions with Applications to Predicate Encryption. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 67–98. Springer, Heidelberg, August 2017. doi:10.1007/978-3-319-63688-7_3.

12 Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012.

13 Boaz Barak, Iftach Haitner, Dennis Hofheinz, and Yuval Ishai. Bounded Key-Dependent Message Security. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 423–444. Springer, Heidelberg, May/June 2010. doi:10.1007/978-3-642-13190-5_22.

14 Amos Beimel, Ariel Gabizon, Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, and Anat Paskin-Cherniavsky. Non-Interactive Secure Multiparty Computation. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 387–404. Springer, Heidelberg, August 2014. doi:10.1007/978-3-662-44381-1_22.

15 Fabrice Benhamouda and Huijia Lin. k-Round Multiparty Computation from k-Round Oblivious Transfer via Garbled Interactive Circuits. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 500–532. Springer, Heidelberg, April/May 2018. doi:10.1007/978-3-319-78375-8_17.

16 John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-Policy Attribute-Based Encryption. In *2007 IEEE Symposium on Security and Privacy*, pages 321–334. IEEE Computer Society Press, May 2007. doi:10.1109/SP.2007.11.

17 Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability Obfuscation from Functional Encryption. In Venkatesan Guruswami, editor, *56th FOCS*, pages 171–190. IEEE Computer Society Press, October 2015. doi:10.1109/FOCS.2015.20.

18 Dan Boneh and Matthew K. Franklin. Identity-Based Encryption from the Weil Pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, Heidelberg, August 2001. doi:10.1007/3-540-44647-8_13.

19 Dan Boneh and Matthew K. Franklin. Identity-Based Encryption from the Weil Pairing. In *CRYPTO*, pages 213–229, 2001. doi:10.1007/3-540-44647-8_13.

20 Dan Boneh, Amit Sahai, and Brent Waters. Functional Encryption: Definitions and Challenges. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 253–273. Springer, Heidelberg, March 2011. doi:10.1007/978-3-642-19571-6_16.

21 Dan Boneh and Brent Waters. Conjunctive, Subset, and Range Queries on Encrypted Data. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 535–554. Springer, Heidelberg, February 2007. doi:10.1007/978-3-540-70936-7_29.

22 Xavier Boyen and Brent Waters. Anonymous Hierarchical Identity-Based Encryption (Without Random Oracles). In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 290–307. Springer, Heidelberg, August 2006. doi:10.1007/11818175_17.

23 Zvika Brakerski, Ilan Komargodski, and Gil Segev. Multi-input Functional Encryption in the Private-Key Setting: Stronger Security from Weaker Assumptions. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 852–880. Springer, Heidelberg, May 2016. doi:10.1007/978-3-662-49896-5_30.

24 Zvika Brakerski and Renen Perlman. Lattice-Based Fully Dynamic Multi-key FHE with Short Ciphertexts. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 190–213. Springer, Heidelberg, August 2016. doi:10.1007/978-3-662-53018-4_8.

25 David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai Trees, or How to Delegate a Lattice Basis. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 523–552. Springer, Heidelberg, May/June 2010. doi:10.1007/978-3-642-13190-5_27.

26 Nishanth Chandran, Vipul Goyal, Aayush Jain, and Amit Sahai. Functional Encryption: Decentralised and Delegatable. *IACR Cryptology ePrint Archive*, 2015:1017, 2015. URL: http://dblp.uni-trier.de/db/journals/iacr/iacr2015.html#ChandranGJS15.

27 Jérémy Chotard, Edouard Dufour Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. Decentralized Multi-Client Functional Encryption for Inner Product. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, LNCS, pages 703–732. Springer, Heidelberg, December 2018. doi:10.1007/978-3-030-03329-3_24.

28 Michael Clear and Ciaran McGoldrick. Multi-identity and Multi-key Leveled FHE from Learning with Errors. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 630–656. Springer, Heidelberg, August 2015. doi:10.1007/978-3-662-48000-7_31.

29 Clifford Cocks. An Identity Based Encryption Scheme Based on Quadratic Residues. In *IMA Int. Conf.*, volume 2260 of *Lecture Notes in Computer Science*, pages 360–363. Springer, 2001.

30 Nico Döttling and Sanjam Garg. Identity-Based Encryption from the Diffie-Hellman Assumption. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 537–569. Springer, Heidelberg, August 2017. doi:10.1007/978-3-319-63688-7_18.

31 Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In *26th ACM STOC*, pages 554–563. ACM Press, May 1994. doi:10.1145/195058.195408.

32 Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-Round Secure MPC from Indistinguishability Obfuscation. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 74–94. Springer, Heidelberg, February 2014. doi:10.1007/978-3-642-54242-8_4.

33 Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013. doi:10.1109/FOCS.2013.13.

34 Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. Attribute-Based Encryption for Circuits from Multilinear Maps. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 479–499. Springer, Heidelberg, August 2013. doi:10.1007/978-3-642-40084-1_27.

35 Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, and Ahmadreza Rahimi. Registration-Based Encryption: Removing Private-Key Generator from IBE. In *TCC 2018, Part I*, LNCS, pages 689–718. Springer, Heidelberg, March 2018. doi:10.1007/978-3-030-03807-6_25.

36 Sanjam Garg, Yuval Ishai, and Akshayaram Srinivasan. Two-Round MPC: Information-Theoretic and Black-Box. In *TCC 2018, Part I*, LNCS, pages 123–151. Springer, Heidelberg, March 2018. doi:10.1007/978-3-030-03807-6_5.

37 Sanjam Garg, Mohammad Mahmoody, and Ameer Mohammed. Lower Bounds on Obfuscation from All-or-Nothing Encryption Primitives. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 661–695. Springer, Heidelberg, August 2017. doi:10.1007/978-3-319-63688-7_22.

38 Sanjam Garg, Mohammad Mahmoody, and Ameer Mohammed. When Does Functional Encryption Imply Obfuscation? In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 82–115. Springer, Heidelberg, November 2017. doi:10.1007/978-3-319-70500-2_4.

39 Sanjam Garg, Peihan Miao, and Akshayaram Srinivasan. Two-Round Multiparty Secure Computation Minimizing Public Key Operations. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 273–301. Springer, Heidelberg, August 2018. doi:10.1007/978-3-319-96878-0_10.

40 Sanjam Garg, Pratyay Mukherjee, Omkant Pandey, and Antigoni Polychroniadou. The Exact Round Complexity of Secure Computation. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 448–476. Springer, Heidelberg, May 2016. doi:10.1007/978-3-662-49896-5_16.

41 Sanjam Garg and Akshayaram Srinivasan. Garbled Protocols and Two-Round MPC from Bilinear Maps. In *58th FOCS*, pages 588–599. IEEE Computer Society Press, 2017. doi:10.1109/FOCS.2017.60.

42 Sanjam Garg and Akshayaram Srinivasan. Two-Round Multiparty Secure Computation from Minimal Assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 468–499. Springer, Heidelberg, April/May 2018. doi:10.1007/978-3-319-78375-8_16.

43 Romain Gay. Personal Communication, 2019.

44 Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008. doi:10.1145/1374376.1374407.

45 Shafi Goldwasser, S. Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input Functional Encryption. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 578–602. Springer, Heidelberg, May 2014. doi:10.1007/978-3-642-55220-5_32.

46 Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 555–564. ACM Press, June 2013. doi:10.1145/2488608.2488678.

47 Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate Encryption for Circuits from LWE. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 503–523. Springer, Heidelberg, August 2015. doi:10.1007/978-3-662-48000-7_25.

48 S. Dov Gordon, Feng-Hao Liu, and Elaine Shi. Constant-Round MPC with Fairness and Guarantee of Output Delivery. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 63–82. Springer, Heidelberg, August 2015. doi:10.1007/978-3-662-48000-7_4.

49 Vipul Goyal. Reducing Trust in the PKG in Identity Based Cryptosystems. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 430–447. Springer, Heidelberg, August 2007. doi:10.1007/978-3-540-74143-5_24.

50 Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 06*, pages 89–98. ACM Press, October/November 2006. Available as Cryptology ePrint Archive Report 2006/309. doi:10.1145/1180405.1180418.

51 Shai Halevi, Yuval Ishai, Abhishek Jain, Ilan Komargodski, Amit Sahai, and Eylon Yogev. Non-Interactive Multiparty Computation Without Correlated Randomness. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 181–211. Springer, Heidelberg, December 2017. doi:10.1007/978-3-319-70700-6_7.

52 Shai Halevi, Yuval Ishai, Abhishek Jain, Ilan Komargodski, Amit Sahai, and Eylon Yogev. Non-Interactive Multiparty Computation without Correlated Randomness. Cryptology ePrint Archive, Report 2017/871, 2017. URL: http://eprint.iacr.org/2017/871.

53 Shai Halevi, Yuval Ishai, Abhishek Jain, Eyal Kushilevitz, and Tal Rabin. Secure Multiparty Computation with General Interaction Patterns. In Madhu Sudan, editor, *ITCS 2016*, pages 157–168. ACM, January 2016. doi:10.1145/2840728.2840760.

54 Jonathan Katz, Amit Sahai, and Brent Waters. Predicate Encryption Supporting Disjunctions, Polynomial Equations, and Inner Products. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 146–162. Springer, Heidelberg, April 2008. doi:10.1007/978-3-540-78967-3_9.

55 Allison B. Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully Secure Functional Encryption: Attribute-Based Encryption and (Hierarchical) Inner Product Encryption. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 62–91. Springer, Heidelberg, May/June 2010. doi:10.1007/978-3-642-13190-5_4.

56 Huijia Lin. Indistinguishability Obfuscation from SXDH on 5-Linear Maps and Locality-5 PRGs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 599–629. Springer, Heidelberg, August 2017. doi:10.1007/978-3-319-63688-7_20.

57 Pratyay Mukherjee and Daniel Wichs. Two Round Multiparty Computation via Multi-key FHE. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 735–763. Springer, Heidelberg, May 2016. doi:10.1007/978-3-662-49896-5_26.

58 Adam O'Neill. Definitional Issues in Functional Encryption. Cryptology ePrint Archive, Report 2010/556, 2010. URL: http://eprint.iacr.org/2010/556.

59 Chris Peikert and Sina Shiehian. Multi-key FHE from LWE, Revisited. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 217–238. Springer, Heidelberg, October/November 2016. doi:10.1007/978-3-662-53644-5_9.

60 Phillip Rogaway. The Moral Character of Cryptographic Work, December 2015. URL: http://web.cs.ucdavis.edu/~rogaway/papers/moral.html.

61 Amit Sahai and Brent R. Waters. Fuzzy Identity-Based Encryption. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 457–473. Springer, Heidelberg, May 2005. doi:10.1007/11426639_27.

62 Brent Waters. Functional Encryption for Regular Languages. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 218–235. Springer, Heidelberg, August 2012. doi:10.1007/978-3-642-32009-5_14.

A Additional Related Work

Functional Encryption

FE started with the notion of "attribute-based encryption" [61, 50] and evolved over time to a more general primitive that encompasses several primitives such as (hierarchical) identity based encryption [19, 29, 22, 44, 25, 4], attribute based encryption [61, 50, 16, 34], predicate encryption [21, 47, 54, 55, 5, 62, 47] and reusable garbled circuits [46]. Formal definitions of the general primitive were first given in [20, 58]. While there has been substantial progress in constructing FE from standard assumptions [1, 7, 56], the general notion of FE for arbitrary polynomial sized circuits was constructed in the breakthrough work of [33] from indistinguishability obfuscation (iO) [12, 33, 56]. Functional encryption for restricted functionalities such as inner products [1, 7], and quadratic functions [56, 11] from more standard assumptions has also been developed.

Multi-Input Functional Encryption

Extending the more basic concept of functional encryption (FE) [61, 20, 58], the notion of multi-input function encryption (MIFE) was first introduced by Goldwasser et al. [45] and there have since been a number of follow-up works. Ananth and Jain [8] show that private key MIFE for general polynomial-arity functions *implies* iO. On the other hand, Brakerski, Komargodski and Segev [23] construct private-key MIFE for constant-arity functions, based on a private-key single-input FE scheme. They achieve adaptive security but also do not consider sender corruption. Badrinarayanan et al. [10] construct MIFE schemes for "unbounded arity" functions. More recent work [3, 2] constructs *inner-product* MIFE.

Multi-Party Computation

Traditional MPC is *interactive*. Ad hoc MIFE can be seen as a special form of *non-interactive* MPC [31, 14, 53]. In particular, ad hoc MIFE separates inputs and functions, which affords greater flexibility – one can use the same encrypted inputs with different functions, or different encrypted inputs with the same function. Moreover, previous non-interactive MPC protocols require a global setup procedure. In a recent work, [52] constructs non-interactive MPC from indistinguishability obfuscation and DDH, assuming a PKI setup and a CRS, without this requirement. In contrast, our schemes are based on standard MIFE for a given functionality and do not require a CRS in general, as we do not necessarily consider sender corruption.

Multi-Authority Functional Encryption

Our work should also be compared to that of Chandran et al. [26], who proposed a notion of "multi-authority" FE (MAFE). In MAFE, key authorities independently generate their own keys. Roughly speaking, to derive a decryption key for a function f, a user must obtain a partial decryption key for f from each authority. In our context, we could think of the authorities as sources. However, a fundamental difference between multi-authority FE and ad hoc MIFE is that in the former, to encrypt, one needs to know the master public keys of all authorities (users). This is a severe limitation, as a user may not be aware of which other parties are to be involved in a computation at the time of encryption. Furthermore, in multi-authority FE, a given ciphertext can only be used in a computation associated with one fixed group, unlike ad hoc MIFE, where a ciphertext can be used in an unbounded number of dynamically-chosen groups. Finally, in MAFE, decryption only operates on a *single* ciphertext, unlike our notion which is intrinsically multi-user.

B Preliminaries

In this section we define the notation and preliminaries used in our work.

B.1 Notation and Conventions

PPT stands for "probabilistic polynomial time" and PT stands for "polynomial time." Algorithms are PPT unless otherwise noted. Throughout, κ denotes the security parameter and 1^κ its unary encoding. For a probabilistic algorithm $\mathcal{A}$, we denote by $\mathcal{A}(x; r)$ the output of $\mathcal{A}$ on input x with random tape r. We denote $y \leftarrow\!\!\$\, A(x)$ as the process of sampling r at random and letting $y \leftarrow A(x; r)$. For a finite set S, we denote $x \leftarrow\!\!\$\, S$ as the process of sampling x uniformly from S. For a distribution $\mathcal{D}$ we denote $x \leftarrow\!\!\$\, \mathcal{D}$ as the process of sampling x according to $\mathcal{D}$. For $k \in \mathbb{N}$ we let $[k]$ denote the set $\{1, \cdots, k\}$. If s is string then $|s|$ denotes its length and $s[i]$ denotes its i-th bit. If $\mathbf{x}$ is a vector then $|\mathbf{x}|$ denotes its number of components and $\mathbf{x}[i]$ denotes its i-th component. We will use $\mathsf{negl}(\cdot)$ to denote an unspecified negligible function and $\mathsf{poly}(\cdot)$ to denote an unspecified polynomial. We say that (families of) distributions $\{\mathcal{D}_{0,\kappa}\}_{\kappa\in\mathbb{N}}, \{\mathcal{D}_{1,\kappa}\}_{\kappa\in\mathbb{N}}$ are computationally indistinguishable if for all PPT adversaries A, $\Pr[\,A(\mathcal{D}_{0,\kappa}) = 1\,] - \Pr[\,A(\mathcal{D}_{1,\kappa}) = 1\,] = \mathsf{negl}(\kappa)$. We write this $\mathcal{D}_{0,\kappa} \underset{C}{\approx} \mathcal{D}_{1,\kappa}$.

B.2 Two-Round MPC

A *2-round MPC protocol* MPC for message-space $\{\mathcal{M}_\kappa\}_{\kappa\mathbb{N}}$ and functionality $\{\mathcal{F}_\kappa\}_{\kappa\in\mathbb{N}}$ where for each $\kappa \in \mathbb{N}$ each $f \in \mathcal{F}_\kappa$ is a function on $(\mathcal{M}_\kappa)^n$ for some n, consists of three algorithms with the following syntax:

- $\mathsf{RunRoundOne}(1^\kappa, 1^n, f, i, x)$: A PPT algorithm taking the security parameter κ, number of users n, a (description of a) function $f \in \mathcal{F}_\kappa$ of arity n, an index $i \in [n]$, an input $x \in \mathcal{M}_\kappa$, and outputting a first protocol message $\rho^{(1)}$ and secret $\mathfrak{s}$.
- $\mathsf{RunRoundTwo}(\mathfrak{s}, (\rho_1^{(1)}, \ldots, \rho_n^{(1)}))$: A PPT algorithm taking a secret $\mathfrak{s}$ and the first protocol message for all n parties $\rho_1^{(1)}, \ldots, \rho_n^{(1)}$, and outputting a second protocol message $\rho^{(2)}$.
- $\mathsf{ComputeResult}$: A PT algorithm taking as input the n second-round protocol messages $\rho_1^{(2)}, \ldots, \rho_n^{(2)}$ for each party and outputting a value y.

Correctness

We say that MPC is *correct* if for all $\kappa, n, \in \mathbb{N}$, $\mathbf{x}_1 \ldots \mathbf{x}_n \in \mathcal{M}_\kappa$ and $f \in \mathcal{F}_\kappa$

$$\Pr\left[y = f(\mathbf{x}) \left| \begin{array}{ll} (\rho_i^{(1)}, \mathfrak{s}_i) \leftarrow\!\!\$\ \mathsf{RunRoundOne}(1^\kappa, 1^n, f, i, \mathbf{x}_i) & \forall i \in [n] \\ \rho_i^{(2)} \leftarrow\!\!\$\ \mathsf{RunRoundTwo}(\mathfrak{s}_i, (\rho_1^{(1)}, \ldots, \rho_n^{(1)})) & \forall i \in [n] \\ y \leftarrow \mathsf{ComputeResult}(\rho_1^{(2)}, \ldots, \rho_n^{(2)}) & \end{array} \right. \right] = 1\,.$$

▶ Remark 6. The above definition of two-round MPC is without setup (*i.e.*, a CRS). We also consider the case that there is an additional algorithm CRSGen taking 1^κ and outputting a common reference string CRS that is input to the remaining algorithms. We call this two-round MPC *in the CRS model.*

We say that MPC is *unbounded* if the output of $\mathsf{RunRoundOne}$ does not depend on n. In this case, we input n to $\mathsf{RunRoundTwo}$ instead of $\mathsf{RunRoundOne}$. We call MPC *input-delayed* (resp. *function-delayed*) if the output of $\mathsf{RunRoundOne}$ does not depend on x (resp. f) but just on $1^{|x|}$ (resp. $1^{|f|}$). In this case, we input x (resp. f) to $\mathsf{RunRoundTwo}$ instead of $\mathsf{RunRoundOne}$. We call MPC *input-rerunnable* (resp. *function-rerunnable*) if it is *input-delayed* (resp. *function-delayed*) and if $\mathsf{RunRoundTwo}$ can be executed multiple times with different input choices (resp. function choices) while still preserving the security properties of the MPC protocol (see below).

Security

Let MPC be a 2-round MPC protocol as above. Let Coins be the coin-space for the protocol. For an adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ and simulator $\mathcal{S}$, consider the experiments in Figure 5. We say that $\mathcal{A}$ is *passive* (aka. semi-honest) of $I = \emptyset$. We say that MPC is SIM-secure if for any PPT adversary $\mathcal{A}$ there is a stateful PPT simulator $\mathcal{S} = (\widetilde{\mathsf{CRSGen}}, \widetilde{\mathsf{Extract}}, \widetilde{\mathsf{Sim}})$ such that $\mathbf{REAL}_{\mathcal{A}}^{\mathsf{MPC}}(\cdot)$ and $\mathbf{IDEAL}_{\mathcal{A},\mathcal{S}}^{\mathsf{MPC}}(\cdot)$ are computationally indistinguishable. Note that

```
Experiment $\mathbf{REAL}^{\mathsf{MPC}}_{\mathcal{A}}(1^\kappa)$

    ($\mathbf{Optional}$) $\mathsf{crs} \leftarrow\!\!\$\ \mathsf{CRSGen}(1^\kappa)$
    $(1^n, I, f, (x_i)_{i \notin I}) \leftarrow\!\!\$\ \mathcal{A}_0(1^\kappa)$
      // $f \in \mathcal{F}_\kappa$ of arity $n$, $x_i \in \mathcal{M}_\kappa$
    $((\rho_i^{(1)})_{i \in I}, \mathsf{st}) \leftarrow\!\!\$\ \mathcal{A}_1(n, I, f)$
    For $i \notin I$ do:
      $r_i \leftarrow\!\!\$\ \mathsf{Coins}(1^\kappa)$
      $(\rho_i^{(1)}, \mathfrak{s}_i)$
       $\leftarrow\!\!\$\ \mathsf{RunRoundOne}(1^\kappa, 1^n, f, i, x_i; r_i)$
    For $i \notin I$ do:
      $\rho_i^{(2)}$
       $\leftarrow\!\!\$\ \mathsf{RunRoundTwo}(\mathfrak{s}_i, (\rho_1^{(1)}, \ldots, \rho_n^{(1)}); r_i)$
    $\alpha \leftarrow\!\!\$\ \mathcal{A}_2(\mathsf{st}, (\rho_i^{(1)}, \rho_i^{(2)})_{i \notin I})$
    Return $\alpha$
```

```
Experiment $\mathbf{IDEAL}^{\mathsf{MPC}}_{\mathcal{A},\mathcal{S}}(1^\kappa)$

    ($\mathbf{Optional}$) $\mathsf{crs} \leftarrow\!\!\$\ \widetilde{\mathsf{CRSGen}}(1^\kappa)$
    $(1^n, I, f, (x_i)_{i \notin I}) \leftarrow\!\!\$\ \mathcal{A}_0(1^\kappa)$
      // $f \in \mathcal{F}_\kappa$ of arity $n$, $x_i \in \mathcal{M}_\kappa$
    $((\rho_i^{(1)})_{i \in I}, \mathsf{st}) \leftarrow\!\!\$\ \mathcal{A}_1(n, I, f)$
    $x_i \leftarrow \widetilde{\mathsf{Extract}}(\rho_i^{(1)}) \quad \forall i \in I$
    $(\rho_i^{(1)}, \rho_i^{(2)})_{i \notin I}$
     $\leftarrow\!\!\$\ \widetilde{\mathsf{Sim}}((x_i)_{i \in I}, f(x_1, \ldots, x_n))$
    $\alpha \leftarrow\!\!\$\ \mathcal{A}_2(\mathsf{st}, (\rho_i^{(1)}, \rho_i^{(2)})_{i \notin I})$
    Return $\alpha$
```

Figure 5 Experiments for SIM-security of two-round MPC.

simulation of the first-round protocol messages for the honest parties are independent of the inputs, so for convenience and ease of presentation we will partition the algorithm $\widetilde{\mathsf{Sim}}$ into two algorithms: $\widetilde{\mathsf{Sim}}_1$ and $\widetilde{\mathsf{Sim}}_2$, defined as follows:

- $\widetilde{\mathsf{Sim}}_1() \mapsto (\rho_i^{(1)})_{i \notin I}$: Outputs the first-round protocol messages for the honest parties.
- $\widetilde{\mathsf{Sim}}_2((x_i)_{i \in I}, y) \mapsto (\rho_i^{(2)})_{i \notin I}$: On input the inputs of the corrupted parties along with the target output value of the protocol, $\widetilde{\mathsf{Sim}}_2$ outputs the second-round protocol messages for the honest parties.

Input/Function-Rerunnability

For simplicity, the definition in Figure 5 does not capture input/function-rerunnability. It is straightforward to see how the definition can be extended. For example in the case of function-rerunnability (the situation is analogous for input-rerunnability where inputs and functions are swapped), the changes to the definition are (1) $\mathcal{A}_0$ outputs a set of functions $\{f_i\}$ instead of a single function, (2) in the real experiment $\mathsf{RunRoundTwo}$ is executed for each function, (3) in the ideal experiement $\widetilde{\mathsf{Sim}}_2$ is called for each function, and (4) the complete set of second-round protocol messages for all functions is given to $\mathcal{A}_2$.

Input Extractability

Our results in this work rely on the simulator's ability to extract the inputs of the corrupted parties, hence the need for the $\widetilde{\mathsf{Extract}}$ algorithm. In the semi-honest setting, extraction is not necessary. In the malicious case, both known constructions of two-round MPC [33, 57] for general functions satisfy the above extractability property, albeit in the CRS model.

B.3 Punctured Pseudorandom Functions

A PRF F is specified by two algorithms:

- $\mathsf{PRF.Setup}(1^\kappa)$: The setup algorithm takes as input the security parameter and outputs a description of the key space $\mathcal{K}_\kappa$, domain $\mathcal{X}$, range $\mathcal{Y}$ as well as the PRF key K.
- $\mathsf{PRF.Eval}(K, x)$: The eval algorithm takes a key $K \in \mathcal{K}_\kappa$ and domain point $x \in \mathcal{X}_\kappa$ and outputs a range point $y \in \mathcal{Y}_\kappa$.

We require that for all adversaries $\mathcal{A}$

$$\Pr\left[\mathcal{A}^{\mathsf{PRF.Eval}(K,\cdot)} \text{ outputs } 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot)} \text{ outputs } 1\right]$$

is negligible in κ, where $K \leftarrow \mathsf{PRF.Setup}(1^\kappa)$ and $\$(\cdot)$ denotes a random function from $\mathcal{X}_\kappa$ to $\mathcal{Y}_\kappa$.

A punctureable PRF additionally includes an algorithm $\mathsf{PRF.Punc}$ which takes as input a PRF key K and a point $x^* \in \mathcal{X}$ and outputs a punctured key K_{x^*}. For correctness, we require that $\mathsf{PRF.Eval}(K_{x^*}, x) = \mathsf{PRF.Eval}(K, x)$ for all $x \neq x^*$ and $\perp$ when $x = x^*$.

Security of punctured PRF

The security game between the challenger and the adversary A consists of the following four phases.

Setup Phase: The challenger samples a PRF key K and a random bit b.

Evaluation Query Phase: The adversary $\mathcal{A}$ queries for polynomially many evaluations. For each evaluation query x, the challenger sends $\mathsf{F}(K, x)$ to $\mathcal{A}$.

Challenge Phase: $\mathcal{A}$ chooses a challenge x^* and the challenger computes $K_{x^*} \leftarrow \mathsf{PRF.Punc}(K, x^*)$. If $b = 0$, the challenger outputs K_{x^*} and $\mathsf{F}(K, x^*)$. Else, the challenger outputs K_{x^*} and $y \leftarrow_{\$} \mathcal{Y}$ chosen uniformly at random.

Guess: The adversary $\mathcal{A}$ outputs a guess b' of b.

The adversary $\mathcal{A}$ wins if $b' = b$ and the adversary did not query for evaluation on x^*. The advantage of $\mathcal{A}$ is defined to be

$$\mathsf{Adv}^{\mathsf{F}}_{\mathcal{A}}(1^\kappa) = |\Pr[\mathcal{A} \text{ wins}] - 1/2|$$

The PRF F is a secure puncturable PRF if for all probabilistic polynomial time adversaries $\mathcal{A}$, we have that $\mathsf{Adv}^{\mathsf{F}}_{\mathcal{A}}(1^\kappa)$ is negligible in κ.

B.4 Multi-Input Functional Encryption

An *n-input FE scheme* [45] MIFE for a message space $\{\mathcal{M}_\kappa\}_{\kappa\in\mathbb{N}}$ and a functionality $\{\mathcal{F}_\kappa\}_{\kappa\in\mathbb{N}}$, where for each $\kappa \in \mathbb{N}$, each $f \in \mathcal{F}_\kappa$ is a (description of a) function on $(\mathcal{M}_\kappa)^n$, is given by a set of algorithms with the following syntax:

- $\mathsf{MIFE.Setup}(1^\kappa, 1^n)$: A PPT algorithm taking the security parameter κ and number of users n, and outputting the master secret key MSK and encryption keys $(\mathsf{EK}_1, \ldots, \mathsf{EK}_n)$.
- $\mathsf{MIFE.KeyGen}(\mathsf{MSK}, f)$: A PT algorithm taking a master secret key MSK, a function $f \in \mathcal{F}_\kappa$ and outputting a corresponding decryption key DK_f.
- $\mathsf{MIFE.Enc}(\mathsf{EK}, \mathbf{x})$: A PPT algorithm taking an encryption key EK and a message $\mathbf{x} \in \mathcal{M}_\kappa$, and outputting a ciphertext c.
- $\mathsf{MIFE.Dec}(\mathsf{DK}_f, (c_1, \ldots, c_n))$: A PT algorithm taking decryption key DK_f and vector of ciphertexts $(c_1, \ldots, c_n)$, and outputting a string y.

Correctness

We say that MIFE is *correct* if for all $\kappa, \in \mathbb{N}$, $\mathbf{x}_1 \dots \mathbf{x}_n \in \mathcal{M}_\kappa$ and $f \in \mathcal{F}_\kappa$

$$\Pr\left[y = f(\mathbf{x}_1, \cdots, \mathbf{x}_n) \left| \begin{array}{l} ((\mathsf{EK}_1, \dots, \mathsf{EK}_n), \mathsf{MSK}) \leftarrow\!\!\$\ \mathsf{MIFE.Setup}(1^\kappa) \\ c_i \leftarrow\!\!\$\ \mathsf{MIFE.Enc}(\mathsf{EK}_i, \mathbf{x}_i) \quad \forall i \in [n] \\ \mathsf{DK}_f \leftarrow\!\!\$\ \mathsf{MIFE.KeyGen}(\mathsf{MSK}, f) \\ y \leftarrow \mathsf{MIFE.Dec}(\mathsf{DK}_f, (c_1, \dots, c_n)) \end{array} \right. \right] = 1 \, .$$

▶ Remark 7. We remark that our formulation of MIFE assumes that the senders (as in our application an encryptor is referred to as a sender or source) are ordered. This allows for consistency with our formulation of ad hoc MIFE and allows us to simplify exposition versus [45].

Indistinguishability-Based Security

For an n-input FE scheme MIFE as above and adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$, consider the experiment in Figure 6.

```
Experiment $\mathbf{IND}^{\mathsf{MIFE}}_{\mathcal{A}}(1^\kappa)$
    $(I, st) \leftarrow\!\!\$\ \mathcal{A}_0(1^\kappa)$
    $b \leftarrow\!\!\$\ \{0,1\}$
    $((\mathsf{EK}_1, \dots, \mathsf{EK}_n), \mathsf{MSK})$
     $\leftarrow\!\!\$\ \mathsf{MIFE.Setup}(1^\kappa)$
    $st \leftarrow\!\!\$\ \mathcal{A}_1^{\mathcal{O}_{\mathsf{enc}}(\cdot,\cdot,\cdot),\mathcal{O}_{\mathsf{kg}}(\cdot)}(st)$
    $b' \leftarrow\!\!\$\ \mathcal{A}_2^{\mathcal{O}_{\mathsf{enc}}(\cdot,\cdot,\cdot),\mathcal{O}_{\mathsf{kg}}(\cdot)}((\mathsf{EK}_i)_{i \in I}, st)$
    Return $(b = b')$

Oracle $\mathcal{O}_{\mathsf{enc}}(i, \mathbf{x}_0, \mathbf{x}_1)$
    If $\mathbf{x}_0, \mathbf{x}_1 \in \mathcal{M}_\kappa$ and $|\mathbf{x}_0| = |\mathbf{x}_1|$
        $c_b \leftarrow\!\!\$\ \mathsf{MIFE.Enc}(\mathsf{EK}_i, \mathbf{x}_b)$
        Return $c_b$
    Else Return $\perp$

Oracle $\mathcal{O}_{\mathsf{kg}}(f)$
    If $f \in \mathcal{F}_\kappa$
        $\mathsf{DK}_f \leftarrow\!\!\$\ \mathsf{MIFE.KeyGen}(\mathsf{MSK}, f)$
        Return $\mathsf{DK}_f$
    Else return $\perp$
```

Figure 6 Experiment for IND-security of standard MIFE.

We call $\mathcal{A}$ *legitimate* if for all $\kappa \in \mathbb{N}$, in all transcripts $\mathbf{IND}^{\mathsf{MIFE}}_{\mathcal{A}}(1^\kappa)$ it holds that for every key generation query f there does not exist two sequences $(y_{1,0}, \cdots, y_{n,0})$ and $(y_{1,1}, \cdots, y_{n,1})$ such that

$$f(y_{1,0}, \cdots, y_{n,0}) \neq f(y_{1,1}, \cdots, y_{n,1})$$

and for every $j \in [n]$

- $j \in I$, *i.e.* j is corrupted (so there is no restriction on $y_{j,0}, y_{j,1}$ above), or
- there is an encryption query $(j, \mathbf{x}_0, \mathbf{x}_1)$ such that $y_{j,0} = \mathbf{x}_0$ and $y_{j,1} = \mathbf{x}_1$.

We assume adversaries are legitimate unless otherwise stated. We call $\mathcal{A}$ *passive* if $I = \emptyset$. We call $\mathcal{A}$ *selective* if $\mathcal{A}_2$ makes no queries. We say that MIFE is IND-secure if for any adversary $\mathcal{A}$

$$|\Pr\left[\mathbf{IND}^{\mathsf{MIFE}}_{\mathcal{A}}(\cdot) \text{ outputs } 1\right] - 1/2| = \mathsf{negl}(\cdot) \, .$$

Simulation-Based Security

For an MIFE scheme MIFE as above, adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$, and simulator $\mathcal{S} = (\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Enc}}, \widetilde{\mathsf{KeyGen}})$, consider the experiments in Figure 7. Here $q_{\mathsf{enc},i}$ is the number of encryption queries for user $i \in [n]$. We say that $\mathcal{A}$ is *q-bounded* if $q_{\mathsf{enc},i} \leq q$ for all $i \in [n]$. We say that MIFE is SEL-SIM-secure if for any adversary $\mathcal{A}$ there is a simulator $\mathcal{S}$ such that $\mathbf{REAL}^{\mathsf{MIFE}}_{\mathcal{A}}(\cdot)$ and $\mathbf{IDEAL}^{\mathsf{MIFE}}_{\mathcal{A},\mathcal{S}}(\cdot)$ are computationally indistinguishable.

▶ Remark 8. For consistency with our formulation of simulation-based security for ad hoc MIFE, we restrict the above definition to the selective-security and passive (no sender corruption) setting.

```
Experiment $\mathbf{REAL}^{\mathsf{MIFE}}_{\mathcal{A}}(1^\kappa)$
    $((\mathsf{EK}_1, \ldots, \mathsf{EK}_n), \mathsf{MSK}) \leftarrow\!\$\ \mathsf{MIFE.Setup}(1^\kappa)$
    $(m_{i,j})_{i\in[n],j\in[q_{\mathsf{enc},i}]} \leftarrow\!\$\ \mathcal{A}_1(1^\kappa)$
    For all $i \in [n], j \in [q_{\mathsf{enc},i}]$ do:
        $c_{i,j} \leftarrow\!\$\ \mathsf{MIFE.Enc}(\mathsf{MSK}_i, m_{i,j})$
    $\alpha \leftarrow\!\$\ \mathcal{A}_2^{\mathcal{O}_{\mathsf{kg}}(\cdot)}((\mathsf{EK}_i)_{i\in[n]}, (c_{i,j})_{i\in[n],j\in[q_{\mathsf{enc},i}]})$
    Return $((m_{i,j})_{i\in[n],j\in[q_{\mathsf{enc},i}]}, \alpha)$

Experiment $\mathbf{IDEAL}^{\mathsf{MIFE}}_{\mathcal{A},\mathcal{S}}(1^\kappa)$
    $((\widetilde{\mathsf{EK}}_1, \ldots, \widetilde{\mathsf{EK}}_n), \widetilde{\mathsf{MSK}}) \leftarrow\!\$\ \widetilde{\mathsf{Setup}}(1^\kappa)$
    $(m_{i,j})_{i\in[n],j\in[q_{\mathsf{enc},i}]} \leftarrow\!\$\ \mathcal{A}_1(st)$
    For all $i \in [n], j \in [q_{\mathsf{enc},i}]$ do:
        $\widetilde{c_{i,j}} \leftarrow\!\$\ \widetilde{\mathsf{Enc}}(\widetilde{\mathsf{MSK}}, |m_{i,j}|)$
    $\alpha \leftarrow\!\$\ \mathcal{A}_2^{\mathcal{O}_{\mathsf{kg}}(\cdot,\cdot,\cdot)}((\widetilde{\mathsf{EK}}_i)_{i\in[n]}, (\widetilde{c_{i,j}}))$
    Return $((m_{i,j})_{i\in[n],j\in[q_{\mathsf{enc},i}]}, \alpha)$

Oracle $\mathcal{O}_{\mathsf{kg}}(f)$   // $f \in \mathcal{F}_\kappa$
    $\mathsf{DK}_f \leftarrow\!\$\ \mathsf{MIFE.KeyGen}(\mathsf{MSK}, f)$
    Return $\mathsf{DK}_f$

Oracle $\widetilde{\mathcal{O}}_{\mathsf{kg}}(f)$   // $f \in \mathcal{F}_\kappa$
    $\widetilde{\mathsf{DK}}_f \leftarrow$
        $\widetilde{\mathsf{KeyGen}}(f, (f(m_{i,j_i}))_{j_i\in[q_{\mathsf{enc},i}]}, \widetilde{\mathsf{MSK}})$
    Return $\widetilde{\mathsf{DK}}_f$
```

Figure 7 Experiments for SIM-security of standard MIFE.

B.5 Function-Private Functional Encryption

A functional encryption scheme, denoted as FE [20], is a tuple of algorithms $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.KeyGen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$ for a message space $\{\mathcal{M}_\kappa\}_{\kappa\in\mathbb{N}}$ and a functionality $\{\mathcal{F}_\kappa\}_{\kappa\in\mathbb{N}}$, where for each $\kappa \in \mathbb{N}$, each $f \in \mathcal{F}_\kappa$ is a (description of a) function on $\mathcal{M}_\kappa$. The syntax is the same as for a 1-input MIFE scheme where $\mathsf{EK}_1 = \mathsf{MSK}$ and $I = \emptyset$. The correctness requirement remains the same, as well the notion of indistinguishability based security (which we refer to as "message privacy").

Function Privacy

We additionally define the notion of function privacy as follows. For an FE scheme FE as above and adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$, consider the experiment in Figure 8.

We call $\mathcal{A}$ *legitimate* if for all $\kappa \in \mathbb{N}$, in all transcripts $\mathbf{FP}^{\mathsf{FE}}_{\mathcal{A}}(1^\kappa)$ it holds that for every key generation query f_0, f_1 there does not exist an encryption query $\mathbf{x} \in \mathcal{M}_\kappa$ such that

$$f_0(\mathbf{x}) \neq f_1(\mathbf{x}) \ .$$

We assume adversaries are legitimate unless otherwise stated. We say that FE is FP-secure if for any adversary $\mathcal{A}$

$$|\Pr\left[\mathbf{FP}^{\mathsf{FE}}_{\mathcal{A}}(\cdot) \text{ outputs } 1\right] - 1/2| = \mathsf{negl}(\cdot) \ .$$

```
Experiment $\mathbf{FP}^{\mathsf{FE}}_{\mathcal{A}}(1^\kappa)$
    $b \leftarrow\$ \{0,1\}$
    $\mathsf{MSK} \leftarrow\$ \mathsf{FE.Setup}(1^\kappa)$
    $b' \leftarrow\$ \mathcal{A}_2^{\mathcal{O}_{\mathsf{enc}}(\cdot),\mathcal{O}_{\mathsf{kg}}(\cdot,\cdot)}(1^\kappa)$
    Return $(b = b')$
```

```
Oracle $\mathcal{O}_{\mathsf{enc}}(\mathbf{x})$
    If $\mathbf{x} \in \mathcal{M}_\kappa$
        $c \leftarrow\$ \mathsf{MIFE.Enc}(\mathsf{MSK}, \mathbf{x})$
        Return $c$
    Else Return $\perp$

Oracle $\mathcal{O}_{\mathsf{kg}}(f_0, f_1)$
    If $f_0, f_1 \in \mathcal{F}_\kappa$
        $\mathsf{DK}_{f_b} \leftarrow\$ \mathsf{MIFE.KeyGen}(\mathsf{MSK}, f_b)$
        Return $\mathsf{DK}_{f_b}$
    Else return $\perp$
```

Figure 8 Experiment for FP-security of FE.

C Simulation Based Security for Ad Hoc MIFE

We now present a simulation-based definition of security. The definition has a number of restrictions that we justify below.

For an ad hoc MIFE scheme aMIFE as above, adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$, and simulator $\mathcal{S} = (\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Enc}}, \widetilde{\mathsf{KeyGen}})$, consider the experiments in Figure 9. Here $q_{E,i}$ is the number of encryption queries for user $i \in [n]$. We say that $\mathcal{A}$ is *q-bounded* if $q_{E,i} \leq q$ for all $i \in [n]$.

```
Experiment $\mathbf{REAL}^{\mathsf{aMIFE}}_{\mathcal{A}}(1^\kappa)$
    $(1^n, st) \leftarrow\$ \mathcal{A}_0(1^\kappa)$
    For all $i \in [n]$ do:
        $(\mathsf{MSK}_i, \mathsf{PP}_i) \leftarrow\$ \mathsf{aMIFE.Setup}(1^\kappa)$
    $(m_{i,j})_{i\in[n],j\in[q_{E,i}]} \leftarrow\$ \mathcal{A}_1(st)$
    For all $i \in [n], j \in [q_{E,i}]$ do:
        $c_{i,j} \leftarrow\$ \mathsf{aMIFE.Enc}(\mathsf{MSK}_i, m_{i,j})$
    $\alpha \leftarrow\$ \mathcal{A}_2^{\mathcal{O}_{\mathsf{kg}}(\cdot,\cdot,\cdot)}((\mathsf{PP}_i)_{i\in[n]}, (c_{i,j}))$
    Return $(1^n, (m_{i,j})_{i\in[n],j\in[q_{E,i}]}, \alpha)$

Experiment $\mathbf{IDEAL}^{\mathsf{aMIFE}}_{\mathcal{A},\mathcal{S}}(1^\kappa)$
    $(1^n, st) \leftarrow\$ \mathcal{A}_0(1^\kappa)$   // $I \subset [n]$
    For all $i \in [n]$ do:
        $(\mathsf{MSK}_i, \mathsf{PP}_i) \leftarrow\$ \widetilde{\mathsf{Setup}}(1^\kappa)$
    $(m_{i,j})_{i\in[n],j\in[q_{E,i}]} \leftarrow\$ \mathcal{A}_1(st)$
    For all $i \in [n], j \in [q_{E,i}]$ do:
        $\widetilde{c_{i,j}} \leftarrow\$ \widetilde{\mathsf{Enc}}(\mathsf{MSK}_i, |m_{i,j}|)$
    $\alpha \leftarrow\$ \mathcal{A}_2^{\widetilde{\mathcal{O}}_{\mathsf{kg}}(\cdot,\cdot,\cdot)}((\mathsf{PP}_i)_{i\in[n]}, (\widetilde{c_{i,j}}))$
    Return $(1^n, (m_{i,j})_{i\in[n],j\in[q_{E,i}]}, \alpha)$
```

```
Oracle $\mathcal{O}_{\mathsf{kg}}(i, (i_1, \ldots, i_\ell), f)$
    If $i \in \{i_1, \cdots, i_\ell\}$, and $f \in \mathcal{F}_\kappa$ of arity $\ell$
        $\mathsf{PDK}_{i,f} \leftarrow$
            $\mathsf{aMIFE.KeyGen}(i, \mathsf{MSK}_i, (\mathsf{PP}_{i_j})_{j\in[\ell]}, f)$
        Return $\mathsf{PDK}_{i,f}$
    Else return $\perp$

Oracle $\widetilde{\mathcal{O}}_{\mathsf{kg}}(i, (i_1, \ldots, i_\ell), f)$
    If $i \in \{i_1, \cdots, i_\ell\}$, and $f \in \mathcal{F}_\kappa$ of arity $\ell$
        $q \leftarrow (i, (i_1, \ldots, i_\ell), f)$
        $\mathsf{PDK}_{i,f} \leftarrow$
            $\widetilde{\mathsf{KeyGen}}(q, (f(m_{i_k,j_{i_k}}))_{j_{i_k}\in[q_{E,i_k}]}, (\widetilde{\mathsf{MSK}_{i_j}}))$
        Return $\mathsf{PDK}_{i,f}$
    Else return $\perp$
```

Figure 9 Experiments for SIM-security of ad hoc MIFE.

We say that aMIFE is xxx-SEL-SIM-secure if for any adversary $\mathcal{A}$ of type xxx there is a simulator $\mathcal{S}$ such that $\mathbf{REAL}^{\mathsf{aMIFE}}_{\mathcal{A}}(\cdot)$ and $\mathbf{IDEAL}^{\mathsf{aMIFE}}_{\mathcal{A},\mathcal{S}}(\cdot)$ are computationally indistinguishable.

▶ Remark 9. The above definition has a number of restrictions that we now justify:

- The adversary is passive and does not corrupt any sender. This is justified because otherwise such a scheme implies virtual black-box obfuscation as in the case of standard MIFE [45], which is impossible [12].

- The adversary is selective and chooses its challenge messages before seeing the public parameters of the users. We focus on this formulation for simplicity and leave the study of adaptive security for ad hoc MIFE in the case of simulation-based security for future work.
- The simulator is black-box. This is for simplicity as it is stronger than allowing non-black-box simulation and our constructions achieve it.

D Proof of Theorem 5

▶ **Theorem 5.** *If* MIFE *is a IND-secure MIFE scheme and* MPC *is a SIM-secure 2-round MPC protocol, then our construction is* sel*-IND-secure.*

Proof. The proof of security makes use of a trapdoor data structure which is defined in Figure 10.

The trapdoor data structure

Here, mode is used to indicate whether we are in the real mode Real or trapdoor mode Trap. CT indicates the hardwired MIFE CT which must be output if the field index equals the counter ctr set in the FE key. The fields val_0 and val_1 are used to indicate the values corresponding to bit 0 and bit 1 respectively, where the latter is used when $\mathsf{index} > \mathsf{ctr}$ and the former when $\mathsf{index} < \mathsf{ctr}$.

mode	CT	index	val_0	val_1

Figure 10 Data Structure Trap used for Proof.

The Hybrids

We prove the theorem via a hybrid argument. We describe our hybrids below.

Hybrid 0: This is the real game in which on every encryption query $(i, \mathbf{x}_0, \mathbf{x}_1)$, $\mathbf{x}_0$ is encrypted.

Suppose there are Q_c encryption queries (made selectively). For each $k \in [Q_c]$, let i be the party index queried, $\mathbf{x}_0$ and $\mathbf{x}_1$ the challenge plaintexts, let T be the tag used during encryption and I be the set of users corrupted by the adversary. We use these definitions in the remainder of the proof.

Hybrid 1: The change in this hybrid is twofold.

1. **Simulate the Public Parameters.** We set the first-round protocol message $\rho_i^{(1)}$ for MPC in the public parameters to the output of the simulator $\widetilde{\mathsf{Sim}}_1$ for each uncorrupted user $i \notin I$.
2. **Simulate the Function Key.** For each key generation query $(i, (i_1, \ldots, i_\ell), f)$, we do the following.
   a. Let $J \triangleq I \cap \{i_1, \ldots, i_\ell\}$ be the subset of corrupted users and $\bar{J} = \{i_1, \ldots, i_\ell\} \setminus J$ be the subset of honest users.
   b. We use the simulator's $\widetilde{\mathsf{Extract}}$ algorithm to compute $\mathbf{x}_j \leftarrow \widetilde{\mathsf{Extract}}(\rho_j^{(1)})$ for each corrupted party $j \in J$ where $\mathsf{PP}_j = \rho_j^{(1)}$, then compute $y = \mathsf{GenKeys}_f(\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_\ell})$ where $\mathbf{x}_j$ for $j \in \bar{J}$ is (honest) party j's input to the MPC protocol.

c. Compute $(\rho_j^{(2)})_{j\in\bar{J}} \leftarrow \widetilde{\mathsf{Sim}}_2((x_j)_{j\in J}, y)$ and $s \leftarrow \mathsf{PRF.Eval}(K_i, 0 \parallel (\mathsf{PP}_{i_j})_{j\in[\ell]} \| f)$. Return $(\rho_i^{(2)}, s)$.

See Figure 11 for a formal description.

Indistinguishability of the hybrids follows from the SIM-security of the MPC protocol. As we see in Figure 11, the only difference from Hybrid 0 is that the inputs of the corrupt parties are extracted using the $\widetilde{\mathsf{Extract}}$ algorithm and the protocol transcript is generated using the MPC simulator. Hence, an adversary who distinguishes between Hybrids 0 and 1 implies an adversary against the MPC protocol by a standard reduction.

```
Hybrid 1:
    $(1^n, I, (\mathsf{PP}_i)_{i\in I}, st) \leftarrow\$ \mathcal{A}_0(1^\kappa)$
    $b \leftarrow\$ \{0,1\}$
    $\forall i \notin I$:
    $(\mathsf{PP}_{\mathsf{FE}_i}, \mathsf{MSK}_{\mathsf{FE}_i}) \leftarrow\$ \mathsf{FE.Setup}(1^\kappa)$
    $K_i \leftarrow\$ \mathcal{K}_\kappa$
    $K_i^{\mathsf{punc}} \leftarrow\$ \mathcal{K}_\kappa$
    $(\rho_i^{(1)})_{i\notin I} \leftarrow\$ \widetilde{\mathsf{Sim}}_1()$
    $\mathsf{PP}_i \leftarrow \rho_i^{(1)}$

    $st \leftarrow\$ \mathcal{A}_1^{\mathcal{O}_{\mathsf{enc}}(\cdot,\cdot,\cdot),\mathcal{O}_{\mathsf{kg}}(\cdot,\cdot,\cdot)}(st)$
    $b' \leftarrow\$ \mathcal{A}_2^{\mathcal{O}_{\mathsf{enc}}(\cdot,\cdot,\cdot),\mathcal{O}_{\mathsf{kg}}(\cdot,\cdot,\cdot)}((\mathsf{PP}_i)_{i\notin I}, st)$
    Return $(b = b')$
```

```
Oracle $\mathcal{O}_{\mathsf{enc}}(i, \mathbf{x}_0, \mathbf{x}_1)$
    $T \leftarrow\$ \{0,1\}^\kappa$
    Return $\mathsf{FE.Enc}(\mathsf{MSK}_{\mathsf{FE}_i}, (\mathbf{x}_0, T, K_i^{\mathsf{punc}}, \mathsf{Trap}))$
```

```
Oracle $\mathcal{O}_{\mathsf{kg}}(i, (i_1, \ldots, i_\ell), f)$
    $J \leftarrow I \cap \{i_1, \ldots, i_\ell\}$; $\bar{J} \leftarrow \{i_1, \ldots, i_\ell\} \setminus J$
    $(K_j, \mathsf{MSK}_{\mathsf{FE}_j}) \leftarrow \widetilde{\mathsf{Extract}}(\rho_j^{(1)})$   $\forall j \in J$
    $y \leftarrow \mathsf{GenKeys}_f((K_{i_1}, \mathsf{MSK}_{\mathsf{FE}_{i_1}}), \ldots, (K_{i_\ell}, \mathsf{MSK}_{\mathsf{FE}_{i_\ell}})$
    $(\rho_j^{(2)})_{j\in\bar{J}} \leftarrow\$ \widetilde{\mathsf{Sim}}_2((K_j, \mathsf{MSK}_{\mathsf{FE}_j})_{j\in J}, y)$
    $s \leftarrow \mathsf{PRF.Eval}(K_i, 0 \parallel (\mathsf{PP}_{i_j})_{j\in[\ell]} \parallel f)$
    Return $(\rho_i^{(2)}, s)$.
```

Figure 11 Hybrid 1.

Hybrid 2: In this hybrid, we replace the outputs of the PRF on key K_i for the honest users i with uniformly random strings in the function $\mathsf{GenKeys}$ described in Figure 2. For every key query pertaining to parties $(i_1, \ldots, i_\ell)$ and function f and every honest party i, we replace $\mathsf{PRF.Eval}(K_i, k \parallel (\mathsf{PP}_{i_j})_{j\in[\ell]} \parallel f)$ for $k \in \{0,1,2\}$ as in Hybrid 1 with a fresh uniformly random string. The changes are formally described in Figure 12 wherein the algorithm R is used to generate random strings and keep track of those previously generated.

Indistinguishability of Hybrid 1 and Hybrid 2 follows from the security of the PRF. More precisely, a standard argument iterates through sub-hybrids for each honest party, replacing the PRF outputs with uniformly random strings.

Hybrid 3: In this hybrid, we change how y (the target output passed to $\widetilde{\mathsf{Sim2}}$) is generated in each query $((i_1, \ldots, i_\ell), f)$. In this hybrid, y is randomly sampled. Furthermore for a pair $((i_1, \ldots, i_\ell), f)$ that is *fully queried* i.e. a partial decryption query $(i, (i_1, \ldots, i_\ell), f)$ is made for each $i \in \{i_1, \ldots, i_\ell\} \setminus I$, the final partial decryption key that is issued has its masking value, i.e. the second component of the partial decryption key, generated differently. It is generated as

$$s \leftarrow y \oplus S_1 \oplus S_2 \oplus \mathsf{GenKeys}''((r'_1, \mathsf{MSK}_{\mathsf{FE}_1}), \ldots, (r'_\ell, \mathsf{MSK}_{\mathsf{FE}_\ell}))$$

Here S_1 and S_2 are computed so as to satisfy requisite dependencies. Figure 13 captures these changes formally.

```
Hybrid 2:
    $(1^n, I, (\mathsf{PP}_i)_{i\in I}, st) \xleftarrow{\$} \mathcal{A}_0(1^\kappa)$
    $b \xleftarrow{\$} \{0,1\}$
    $\forall i \notin I$:
    $(\mathsf{PP}_{\mathsf{FE}_i}, \mathsf{MSK}_{\mathsf{FE}_i}) \xleftarrow{\$} \mathsf{FE.Setup}(1^\kappa)$
    ~~$K_i \xleftarrow{\$} \mathcal{K}_\kappa$~~
    $K_i^{\mathsf{punc}} \xleftarrow{\$} \mathcal{K}_\kappa$
    $\Gamma \leftarrow \emptyset$
    $(\rho_i^{(1)})_{i\notin I} \xleftarrow{\$} \widetilde{\mathsf{Sim}_1}()$
    $\mathsf{PP}_i \leftarrow \rho_i^{(1)}$

    $st \xleftarrow{\$} \mathcal{A}_1^{\mathcal{O}_{\mathsf{enc}}(\cdot,\cdot,\cdot),\mathcal{O}_{\mathsf{kg}}(\cdot,\cdot,\cdot)}(st)$
    $b' \xleftarrow{\$} \mathcal{A}_2^{\mathcal{O}_{\mathsf{enc}}(\cdot,\cdot,\cdot),\mathcal{O}_{\mathsf{kg}}(\cdot,\cdot,\cdot)}((\mathsf{PP}_i)_{i\notin I}, st)$
    Return $(b = b')$
```

```
Oracle $\mathcal{O}_{\mathsf{enc}}(i, \mathbf{x}_0, \mathbf{x}_1)$
    $T \xleftarrow{\$} \{0,1\}^\kappa$
    Return $\mathsf{FE.Enc}(\mathsf{MSK}_{\mathsf{FE}_i}, (\mathbf{x}_0, T, K_i^{\mathsf{punc}}, \mathsf{Trap}))$
```

```
Oracle $\mathcal{O}_{\mathsf{kg}}(i, (i_1, \dots, i_\ell), f)$
    $J \leftarrow I \cap \{i_1, \dots, i_\ell\}$; $\bar{J} \leftarrow \{i_1, \dots, i_\ell\} \setminus J$
    $(K_j, \mathsf{MSK}_{\mathsf{FE}_j}) \leftarrow \widetilde{\mathsf{Extract}}(\rho_j^{(1)}) \quad \forall j \in J$
    $(\gamma_j^{(0)}, \gamma_j^{(1)}, \gamma_j^{(2)}) \xleftarrow{\$} R_\Gamma(j, (i_1, \dots, i_\ell), f) \quad \forall j \in \bar{J}$
    $\forall m \in \{0,1,2\}, j \in J$:
    $\gamma_j^{(m)} \leftarrow \mathsf{PRF.Eval}(K_j, m \parallel (\mathsf{PP}_{i_k})_{k\in[\ell]} \parallel f)$
    $S \leftarrow \bigoplus_{j\in J\cup\bar{J}} \gamma_j^{(0)}$
    $y \leftarrow S \oplus \mathsf{GenKeys}'(\{(r_k, r_l', \mathsf{MSK}_{\mathsf{FE}_k})\}_{k\in[\ell]})$
    $(\rho_j^{(2)})_{j\in\bar{J}} \xleftarrow{\$} \widetilde{\mathsf{Sim}_2}(K_j, \mathsf{MSK}_{\mathsf{FE}_j})_{j\in J}, y)$
    Return $(\rho_i^{(2)}, s := \gamma_i^{(0)})$
```

```
Algorithm $\mathsf{GenKeys}'(\{(r_k, r_l', \mathsf{MSK}_{\mathsf{FE}_k})\}_{k\in[\ell]})$
    $(\{\mathsf{EK}_k\}_{k\in[\ell]}, \mathsf{MSK}) \leftarrow \mathsf{MIFE.Setup}(1^\kappa; \bigoplus_{k\in[\ell]} r_k)$
    $\mathsf{SK}_f \leftarrow \mathsf{MIFE.KeyGen}(\mathsf{MSK}, f)$
    $\forall i \in [\ell]$:
    $\mathsf{SK}_{\mathsf{FE}_i} \leftarrow \mathsf{FE.KeyGen}(\mathsf{MSK}_{\mathsf{FE}_i}, \mathsf{ReEnc}_{\mathsf{EK}_i,\perp}; r_i')$
    Return $(\mathsf{SK}_f, \mathsf{SK}_{\mathsf{FE}_1}, \dots, \mathsf{SK}_{\mathsf{FE}_\ell})$
```

```
Algorithm $R_\Gamma(i, (i_1, \dots, i_\ell, f)$
    If $(i, (i_1, \dots, i_\ell), f, \gamma^{(0)}, \gamma^{(1)}, \gamma^{(2)}) \in \Gamma$
        Return $(\gamma^{(0)}, \gamma^{(1)}, \gamma^{(2)})$
    Else
        $\gamma^{(j)} \xleftarrow{\$} \{0,1\}^{\mathsf{rp}} \quad \forall j \in \{0,1,2\}$
        // where $\mathsf{rp}$ is the range of the PRF
        $\Gamma \leftarrow \Gamma \cup \{(i, (i_1, \dots, i_\ell), f, \gamma^{(0)}, \gamma^{(1)}, \gamma^{(2)})\}$
        Return $(\gamma^{(0)}, \gamma^{(1)}, \gamma^{(2)})$
```

Figure 12 Hybrid 2.

Hybrid 2 and Hybrid 3 are distributed identically. First, y is distributed uniformly in both hybrids for all group-function pairs $((i_1, \dots, i_\ell), f)$ that are not *fully queried*. In the case of a group-function pair $((i_1, \dots, i_\ell), f)$ that is fully queried, each partial decryption key $(\rho_i^{(2)}, s_i)$ for $i \in \{i_1, \dots, i_\ell\}$ is such that

$$y \oplus \bigoplus_{i\in[\ell]} s_i = \mathsf{GenKeys}''((r_1', \mathsf{MSK}_{\mathsf{FE}_1}), \dots, (r_\ell', \mathsf{MSK}_{\mathsf{FE}_\ell}))$$

This is distributed the same as the output of $\mathsf{GenKeys}'$ in the previous hybrid. Hence the y values are distributed identically in both hybrids.

Hybrid 4: Let Q_k be the number of subset-function pairs that are *fully queried*. In this hybrid, the key generation algorithm keeps track of the query number $\mathsf{ctr} \in [\mathsf{Q_k}]$ in the ad hoc MIFE function keys. In more detail, the algorithm $\mathsf{GenKeys}''$ invoked for query index j for all $j \in [Q_k]$ is modified to invoke ReEnc with parameter $\mathsf{ctr} = \mathsf{j}$ (please refer to Figure 3) instead of $\perp$.

We claim that by function hiding of FE, the two hybrids are indistinguishable. To see this, note that since $\mathsf{mode} = R$ in all the FE ciphertexts, changing the ctr value in the FE key has no effect on the decryption value obtained, since this field is only relevant in the trapdoor mode, i.e. when $\mathsf{mode} = T$. Hence, the decryption values for both keys remain exactly the same. Then, by security of FE, we have that Hybrids 3 and 4 are indistinguishable. The formal reduction is standard, and constructs everything except the FE ciphertexts and FE function keys as in the previous hybrid, which are obtained using the FE challenger.

```
Hybrid 3:
    $(1^n, I, (\mathsf{PP}_i)_{i\in I}, st) \leftarrow\!\!\$\ \mathcal{A}_0(1^\kappa)$
    $b \leftarrow\!\!\$\ \{0,1\}$
    $\forall i \notin I$:
    $K_i^{\mathsf{punc}} \leftarrow\!\!\$\ \mathcal{K}_\kappa$
    $(\mathsf{PP}_{\mathsf{FE}_i}, \mathsf{MSK}_{\mathsf{FE}_i}) \leftarrow\!\!\$\ \mathsf{FE.Setup}(1^\kappa)$
    $\Gamma \leftarrow \emptyset$
    $Y \leftarrow \emptyset$
    $Q \leftarrow \emptyset$
    $(\rho_i^{(1)})_{i\notin I} \leftarrow\!\!\$\ \widetilde{\mathsf{Sim}_1}()$
    $\mathsf{PP}_i \leftarrow \rho_i^{(1)} \quad \forall i \notin I$

    $st \leftarrow\!\!\$\ \mathcal{A}_1^{\mathcal{O}_{\mathsf{enc}}(\cdot,\cdot,\cdot),\mathcal{O}_{\mathsf{kg}}(\cdot,\cdot,\cdot)}(st)$
    $b' \leftarrow\!\!\$\ \mathcal{A}_2^{\mathcal{O}_{\mathsf{enc}}(\cdot,\cdot,\cdot),\mathcal{O}_{\mathsf{kg}}(\cdot,\cdot,\cdot)}((\mathsf{PP}_i)_{i\notin I}, st)$
    Return $(b = b')$

Algorithm $\mathsf{GenKeys}''_f(\{(r'_k, \mathsf{MSK}_{\mathsf{FE}_k})\}_{k\in[\ell]}$
    $((\mathsf{EK}_1, \ldots, \mathsf{EK}_\ell), \mathsf{MSK}) \leftarrow \mathsf{MIFE.Setup}(1^\kappa)$
    $\mathsf{SK}_f \leftarrow \mathsf{MIFE.KeyGen}(\mathsf{MSK}, f)$
    $\forall i \in [\ell]$, do:
    $\mathsf{SK}_{\mathsf{FE}_i} \leftarrow \mathsf{FE.KeyGen}(\mathsf{MSK}_{\mathsf{FE}_i}, \mathsf{ReEnc}_{\mathsf{EK}_i,\perp}; r'_i)$
    Return $(\mathsf{SK}_f, \mathsf{SK}_{\mathsf{FE}_1}, \ldots, \mathsf{SK}_{\mathsf{FE}_\ell})$
```

```
Oracle $\mathcal{O}_{\mathsf{enc}}(i, \mathbf{x}_0, \mathbf{x}_1)$
    $T \leftarrow\!\!\$\ \{0,1\}^\kappa$
    Return $\mathsf{FE.Enc}(\mathsf{MSK}_{\mathsf{FE}_i}, (\mathbf{x}_0, T, K_i^{\mathsf{punc}}, \mathsf{Trap}))$
```

```
Oracle $\mathcal{O}_{\mathsf{kg}}(i, (i_1, \ldots, i_\ell), f)$
    $J \leftarrow I \cap \{i_1, \ldots, i_\ell\}$; $\bar{J} \leftarrow \{i_1, \ldots, i_\ell\} \setminus J$
    $(\alpha_j, r_j, K_j) \leftarrow \mathsf{Extract}(\rho_j^{(1)}) \quad \forall j \in J$
    $y \leftarrow\!\!\$\ R_Y((i_1, \ldots, i_\ell), f)$
    If $((i_1, \ldots, i_\ell), f, \bar{J}') \notin Q$
        $\bar{J}' \leftarrow \emptyset$
    $\bar{J}' \leftarrow \bar{J}' \cup \{i\}$
    $Q \leftarrow Q \cup ((i_1, \ldots, i_\ell), f, \bar{J}')$
    $(\rho_j^{(2)})_{j\in\bar{J}} \leftarrow\!\!\$\ \widetilde{\mathsf{Sim}_2}((\alpha_j, r_j, K_j)_{j\in J}, y)$
    $\forall j \in \bar{J}$:
    $(\gamma_j^{(0)}, \gamma_j^{(1)}, \gamma_j^{(2)}) \leftarrow\!\!\$\ R_\Gamma(j, (i_1, \ldots, i_\ell), f)$
    $\forall m \in \{0,1,2\}, j \in J$,
    $\gamma_j^{(m)} \leftarrow \mathsf{PRF.Eval}(K_j, m \parallel (\mathsf{PP}_{i_k})_{k\in[\ell]} \parallel f)$
    If $\bar{J}' = \bar{J}$ // if fully queried
        $S_1 \leftarrow\!\!\$\ \bigoplus_{j\in\bar{J}, j\neq i} \gamma_j^{(0)}$
        $S_2 \leftarrow\!\!\$\ \bigoplus_{j\in J} \gamma_j^{(0)}$
        $s \leftarrow y \oplus S_1 \oplus S_2 \oplus$
        $\mathsf{GenKeys}''(\{(\gamma_{i_k}^{(2)}, \mathsf{MSK}_{\mathsf{FE}_{i_k}})\}_{k\in[\ell]})$
    Else
        $s \leftarrow \gamma_i^{(0)}$
    Return $(\rho_i^{(2)}, s)$
```

```
Algorithm $R_\Gamma(i, (i_1, \ldots, i_\ell, f)$
    If $(i, (i_1, \ldots, i_\ell), f, \gamma^{(0)}, \gamma^{(1)}, \gamma^{(2)}) \in \Gamma$
        Return $(\gamma^{(0)}, \gamma^{(1)}, \gamma^{(2)})$
    Else
        $\gamma^{(j)} \leftarrow\!\!\$\ \{0,1\}^{\mathsf{rp}} \quad \forall j \in \{0,1,2\}$
        // where rp is the range of the PRF
        $\Gamma \leftarrow \Gamma \cup \{(i, (i_1, \ldots, i_\ell), f, \gamma^{(0)}, \gamma^{(1)}, \gamma^{(2)})\}$
        Return $(\gamma^{(0)}, \gamma^{(1)}, \gamma^{(2)})$
```

```
Algorithm $R_Y((i_1, \ldots, i_\ell, f)$
    If $((i_1, \ldots, i_\ell), f, y) \in Y$
        Return $y$
    Else
        $y \leftarrow\!\!\$\ \{0,1\}^{\mathsf{rp}}$
        $Y \leftarrow Y \cup \{((i_1, \ldots, i_\ell), f, y)\}$
        Return $y$
```

Figure 13 Hybrid 3.

In more detail, we have a series of subhybrids, one for each $i \notin I$, where in Hybrid $4, i$, the change above is made to the function key associated with the FE instance for party i. Let Hybrid $4, 0$ denote Hybrid 3 and let Hybrid $|n \setminus I|$ denote Hybrid 4. We now give the formal reduction to FE function hiding for distinguishing Hybrid $4, i-1$ and Hybrid $4, i$ (for ease of notation, we assume that the indices i are consecutive). The simulator $\mathcal{B}$ is defined as follows. First $\mathcal{B}$ receives the public parameters PP from the FE challenger. Then it receives $(1^\kappa, I, (\mathsf{PP}_j)_{j\in I})$ from the hybrid distinguisher $\mathcal{A}$. Next it runs Step 2 to Step 10 on the left hand side of Figure 14 and passes $(\mathsf{PP}_j)_{j\notin I}$ (see Step 10) to $\mathcal{A}$. It handles encryption and key generation queries as follows. On an encryption query $(i', \mathbf{x}_0, \mathbf{x}_1)$ with $i' \neq i$, the query is handled the same as $\mathcal{O}_{\mathsf{enc}}$ in Figure 14. On an encryption query $(i, \mathbf{x}_0, \mathbf{x}_1)$, a tag $T \leftarrow\!\!\$\ \{0,1\}^\kappa$ is sampled and $\mathcal{B}$ makes a call to the FE encryption oracle with message $(\mathbf{x}_0, T, K_i^{\mathsf{punc}}, \mathsf{Trap} := (\mathsf{mode} := R, \perp, \perp, \perp, \perp))$ and returns the returned ciphertext. Key generation queries are handled as in $\mathcal{O}_{\mathsf{kg}}$ in Figure 14 with one exception, namely the secret keys $\mathsf{SK}_{\mathsf{FE}_{i'}}$ are computed as in $\mathsf{GenKeys}'''$ except for the case $i' = i$; the secret

```
Hybrid 4:
    $(1^n, I, (\mathsf{PP}_i)_{i \in I}, st) \leftarrow\$ \mathcal{A}_0(1^\kappa)$
    $b \leftarrow\$ \{0,1\}$
    $\forall i \notin I$:
    $K_i^{\mathsf{punc}} \leftarrow\$ \mathcal{K}_\kappa$
    $(\mathsf{PP}_{\mathsf{FE}_i}, \mathsf{MSK}_{\mathsf{FE}_i}) \leftarrow\$ \mathsf{FE.Setup}(1^\kappa)$
    $\Gamma \leftarrow \emptyset$
    $Y \leftarrow \emptyset$
    $Q \leftarrow \emptyset$
    $q \leftarrow 0$
    $(\rho_i^{(1)})_{i \notin I} \leftarrow\$ \widetilde{\mathsf{Sim}_1}()$
    $\mathsf{PP}_i \leftarrow \rho_i^{(1)} \quad \forall i \notin I$

    $st \leftarrow\$ \mathcal{A}_1^{\mathcal{O}_{\mathsf{enc}}(\cdot,\cdot,\cdot), \mathcal{O}_{\mathsf{kg}}(\cdot,\cdot,\cdot)}(st)$
    $b' \leftarrow\$ \mathcal{A}_2^{\mathcal{O}_{\mathsf{enc}}(\cdot,\cdot,\cdot), \mathcal{O}_{\mathsf{kg}}(\cdot,\cdot,\cdot)}((\mathsf{PP}_i)_{i \notin I}, st)$
    Return $(b = b')$

Algorithm $\mathsf{GenKeys}_f'''((r_k', \mathsf{MSK}_{\mathsf{FE}_k})_{k \in [\ell]}, \mathsf{ctr})$
    $((\mathsf{EK}_1, \ldots, \mathsf{EK}_\ell), \mathsf{MSK}) \leftarrow$
    $\mathsf{MIFE.Setup}(1^\kappa)$
    $\mathsf{SK}_f \leftarrow \mathsf{MIFE.KeyGen}(\mathsf{MSK}, f)$
    $\forall i \in [\ell]$, do:
    $\mathsf{SK}_{\mathsf{FE}_i} \leftarrow \mathsf{FE.KeyGen}(\mathsf{MSK}_{\mathsf{FE}_i}, \mathsf{ReEnc}_{\mathsf{EK}_i, \mathsf{ctr}}; r_i')$
    Return $(\mathsf{SK}_f, \mathsf{SK}_{\mathsf{FE}_1}, \ldots, \mathsf{SK}_{\mathsf{FE}_\ell})$
```

```
Oracle $\mathcal{O}_{\mathsf{enc}}(i, \mathbf{x}_0, \mathbf{x}_1)$
    $T \leftarrow\$ \{0,1\}^\kappa$
    Return $\mathsf{FE.Enc}(\mathsf{MSK}_{\mathsf{FE}_i}, (\mathbf{x}_0, T, K_i^{\mathsf{punc}}, \mathsf{Trap}))$
```

```
Oracle $\mathcal{O}_{\mathsf{kg}}(i, (i_1, \ldots, i_\ell), f)$
    $J \leftarrow I \cap \{i_1, \ldots, i_\ell\}$; $\bar{J} \leftarrow \{i_1, \ldots, i_\ell\} \setminus J$
    $(\alpha_j, r_j, K_j) \leftarrow \widetilde{\mathsf{Extract}}(\rho_j^{(1)}) \quad \forall j \in J$
    $y \leftarrow\$ R_Y((i_1, \ldots, i_\ell), f)$
    If $((i_1, \ldots, i_\ell), f, \bar{J}') \notin Q$
        $\bar{J}' \leftarrow \emptyset$
    $\bar{J}' \leftarrow \bar{J}' \cup \{i\}$
    $Q \leftarrow Q \cup ((i_1, \ldots, i_\ell), f, \bar{J}')$
    $(\rho_j^{(2)})_{j \in \bar{J}} \leftarrow\$ \widetilde{\mathsf{Sim}_2}((\alpha_j, r_j, K_j)_{j \in J}, y)$
    $\forall j \in \bar{J}$:
    $(\gamma_j^{(0)}, \gamma_j^{(1)}, \gamma_j^{(2)}) \leftarrow\$ R_\Gamma(j, (i_1, \ldots, i_\ell), f)$
    $\forall m \in \{0,1,2\}, j \in J$,
    $\gamma_j^{(m)} \leftarrow \mathsf{PRF.Eval}(K_j, m \parallel (\mathsf{PP}_{i_k})_{k \in [\ell]} \parallel f)$
    If $\bar{J}' = \bar{J}$ // if fully queried
        $q \leftarrow q + 1$
        $S_1 \leftarrow\$ \bigoplus_{j \in \bar{J}, j \neq i} \gamma_j^{(0)}$
        $S_2 \leftarrow\$ \bigoplus_{j \in J} \gamma_j^{(0)}$
        $s \leftarrow y \oplus S_1 \oplus S_2 \oplus$
        $\mathsf{GenKeys}'''(\{(\gamma_{i_k}^{(2)}, \mathsf{MSK}_{\mathsf{FE}_{i_k}})\}_{k \in [\ell]}, q)$
    Else
        $s \leftarrow \gamma_i^{(0)}$
    Return $(\rho_i^{(2)}, s)$
```

```
Algorithm $R_\Gamma(i, (i_1, \ldots, i_\ell, f)$
    If $(i, (i_1, \ldots, i_\ell), f, \gamma^{(0)}, \gamma^{(1)}, \gamma^{(2)}) \in \Gamma$
        Return $(\gamma^{(0)}, \gamma^{(1)}, \gamma^{(2)})$
    Else
        $\gamma^{(j)} \leftarrow\$ \{0,1\}^{\mathsf{rp}} \quad \forall j \in \{0,1,2\}$
        // where rp is the range of the PRF
        $\Gamma \leftarrow \Gamma \cup \{(i, (i_1, \ldots, i_\ell), f, \gamma^{(0)}, \gamma^{(1)}, \gamma^{(2)})\}$
        Return $(\gamma^{(0)}, \gamma^{(1)}, \gamma^{(2)})$
```

```
Algorithm $R_Y((i_1, \ldots, i_\ell, f)$
    If $((i_1, \ldots, i_\ell), f, y) \in Y$
        Return $y$
    Else
        $y \leftarrow\$ \{0,1\}^{\mathsf{rp}}$
        $Y \leftarrow Y \cup \{((i_1, \ldots, i_\ell), f, y)\}$
        Return $y$
```

Figure 14 Hybrid 4.

key $\mathsf{SK}_{\mathsf{FE}_i}$ is obtained by making a call to the FE key generation oracle with functions $(\mathsf{ReEnc}_{i,\perp}, \mathsf{ReEnc}_{i,\mathsf{ctr}})$. If the FE challenger's bit is 0, then $\mathcal{B}$ perfectly simulates Hybrid $4, i-1$ and if the FE challenger's bit is 1, then $\mathcal{B}$ perfectly simulates Hybrid $4, i$.

For $j \in [Q_k]$, we define:

Hybrid $5_{\mathbf{j},\mathbf{1}}$: In this hybrid, we hardwire all the Q_c MIFE CTs that are output by the j^{th} function query in the corresponding single input FE ciphertexts in the field $\mathsf{Trap.CT}$ and set $\mathsf{mode} = T$.

In more detail, for key query j, we generate the encryption keys exactly as in Figure 2. Now, encryptor $i \in [n]$ computes Q_c ciphertexts as follows:

1. For $k \in [Q_c]$, let $r_{i,j,k} \leftarrow \mathsf{PRF.Eval}(K_i^{\mathsf{punc}}, j \parallel \mathsf{EK}_{i,j} \parallel T_{i,k})$
2. For $k \in [Q_c]$, let $\psi_{i,j,k} = \mathsf{MIFE.Enc}(\mathsf{EK}_{i,j}, \mathbf{x}_{i,k}; r_{i,j,k})$

For $k \in [Q_c]$, encryptor $i \in [n]$ sets Trap so as to program it for the j^{th} function query as follows:

$$\mathsf{Trap.mode} = T, \quad \mathsf{Trap.CT} = \psi_{i,j,k}, \quad \mathsf{Trap.index} = j, \quad \mathsf{val}_0 = \mathbf{x}_{0,i,k}, \quad \mathsf{val}_1 = \mathbf{x}_{1,i,k}$$

Please see Figure 15 for the complete description.

```
Hybrid $5_{j,1}$:
    $(1^n, I, (\mathsf{PP}_i)_{i \in I}, st) \leftarrow_\$ \mathcal{A}_0(1^\kappa)$
    $b \leftarrow_\$ \{0,1\}$
    $\forall i \notin I$:
    $K_i^{\mathsf{punc}} \leftarrow_\$ \mathcal{K}_\kappa$
    $(\mathsf{PP}_{\mathsf{FE}_i}, \mathsf{MSK}_{\mathsf{FE}_i}) \leftarrow_\$ \mathsf{FE.Setup}(1^\kappa)$
    $\Gamma \leftarrow \emptyset$
    $Y \leftarrow \emptyset$
    $Q \leftarrow \emptyset$
    $q \leftarrow 0$
    $\mathsf{mkeys} := ((\mathsf{EK}_{1,\mathsf{j}}, \ldots, \mathsf{EK}_{n,\mathsf{j}}), \mathsf{MSK}_\mathsf{j}) \leftarrow_\$ \mathsf{MIFE.Setup}(1^\kappa)$
    $(\rho_i^{(1)})_{i \notin I} \leftarrow_\$ \widetilde{\mathsf{Sim}}_1()$
    $\mathsf{PP}_i \leftarrow \rho_i^{(1)} \quad \forall i \notin I$

    $st \leftarrow_\$ \mathcal{A}_1^{\mathcal{O}_{\mathsf{enc}}(\cdot,\cdot,\cdot), \mathcal{O}_{\mathsf{kg}}(\cdot,\cdot,\cdot)}(st)$
    $b' \leftarrow_\$ \mathcal{A}_2^{\mathcal{O}_{\mathsf{enc}}(\cdot,\cdot,\cdot), \mathcal{O}_{\mathsf{kg}}(\cdot,\cdot,\cdot)}((\mathsf{PP}_i)_{i \notin I}, st)$
    Return $(b = b')$

Algorithm $\mathsf{GenKeys}_f''''((r'_k, \mathsf{MSK}_{\mathsf{FE}_k})_{k \in [\ell]}, \mathsf{ctr}, \mathsf{mkeys})$
    If $\mathsf{ctr} = \mathsf{j}$
        $((\mathsf{EK}_1, \ldots, \mathsf{EK}_\ell), \mathsf{MSK}) \leftarrow \mathsf{mkeys}$
    Else
        $((\mathsf{EK}_1, \ldots, \mathsf{EK}_\ell), \mathsf{MSK}) \leftarrow \mathsf{MIFE.Setup}(1^\kappa)$
    $\mathsf{SK}_f \leftarrow \mathsf{MIFE.KeyGen}(\mathsf{MSK}, f)$
    $\forall i \in [\ell]$, do:
    $\mathsf{SK}_{\mathsf{FE}_i} \leftarrow \mathsf{FE.KeyGen}(\mathsf{MSK}_{\mathsf{FE}_i}, \mathsf{ReEnc}_{\mathsf{EK}_i,\mathsf{ctr}}; r'_i)$
    Return $(\mathsf{SK}_f, \mathsf{SK}_{\mathsf{FE}_1}, \ldots, \mathsf{SK}_{\mathsf{FE}_\ell})$
```

```
Oracle $\mathcal{O}_{\mathsf{enc}}(i, \mathbf{x}_0, \mathbf{x}_1)$
    $T \leftarrow_\$ \{0,1\}^\kappa$
    $r \leftarrow \mathsf{PRF.Eval}(K_i^{\mathsf{punc}}, \mathsf{j} \parallel \mathsf{EK}_{i,\mathsf{j}} \parallel T)$
    $\psi \leftarrow \mathsf{MIFE.Enc}(\mathsf{EK}_{i,\mathsf{j}}, \mathbf{x}_0; r)$
    $\mathsf{Trap} \leftarrow (\mathsf{mode} := \mathsf{Trap}, \mathsf{CT} := \psi,$
    $\mathsf{index} := \mathsf{j}, \mathsf{val}_0 := \mathbf{x}_0, \mathsf{val}_1 := \mathbf{x}_1)$
    Return $\mathsf{FE.Enc}(\mathsf{MSK}_{\mathsf{FE}_i}, (\mathbf{x}_0, T, K_i^{\mathsf{punc}}, \mathsf{Trap}))$
```

```
Oracle $\mathcal{O}_{\mathsf{kg}}(i, (i_1, \ldots, i_\ell), f)$
    $J \leftarrow I \cap \{i_1, \ldots, i_\ell\}$; $\bar{J} \leftarrow \{i_1, \ldots, i_\ell\} \setminus J$
    $(\alpha_j, r_j, K_j) \leftarrow \widetilde{\mathsf{Extract}}(\rho_j^{(1)}) \quad \forall j \in J$
    $y \leftarrow_\$ R_Y((i_1, \ldots, i_\ell), f)$
    If $((i_1, \ldots, i_\ell), f, \bar{J}') \notin Q$
        $\bar{J}' \leftarrow \emptyset$
    $\bar{J}' \leftarrow \bar{J}' \cup \{i\}$
    $Q \leftarrow Q \cup ((i_1, \ldots, i_\ell), f, \bar{J}')$
    $(\rho_j^{(2)})_{j \in \bar{J}} \leftarrow_\$ \widetilde{\mathsf{Sim}}_2((\alpha_j, r_j, K_j)_{j \in J}, y)$
    $\forall j \in \bar{J}$:
    $(\gamma_j^{(0)}, \gamma_j^{(1)}, \gamma_j^{(2)}) \leftarrow_\$ R_\Gamma(j, (i_1, \ldots, i_\ell), f)$
    $\forall m \in \{0,1,2\}, j \in J$,
    $\gamma_j^{(m)} \leftarrow \mathsf{PRF.Eval}(K_j, m \parallel (\mathsf{PP}_{i_k})_{k \in [\ell]} \parallel f)$
    If $\bar{J}' = \bar{J}$ // if fully queried
        $q \leftarrow q + 1$
        $S_1 \leftarrow_\$ \bigoplus_{j \in \bar{J}, j \neq i} \gamma_j^{(0)}$
        $S_2 \leftarrow_\$ \bigoplus_{j \in J} \gamma_j^{(0)}$
        $s \leftarrow y \oplus S_1 \oplus S_2 \oplus$
        $\mathsf{GenKeys}''''(\{(\gamma_{i_k}^{(2)}, \mathsf{MSK}_{\mathsf{FE}_{i_k}})\}_{k \in [\ell]}, q, \mathsf{mkeys})$
    Else
        $s \leftarrow \gamma_i^{(0)}$
    Return $(\rho_i^{(2)}, s)$
```

```
Algorithm $R_\Gamma(i, (i_1, \ldots, i_\ell, f)$
    If $(i, (i_1, \ldots, i_\ell), f, \gamma^{(0)}, \gamma^{(1)}, \gamma^{(2)}) \in \Gamma$
        Return $(\gamma^{(0)}, \gamma^{(1)}, \gamma^{(2)})$
    Else
        $\gamma^{(j)} \leftarrow_\$ \{0,1\}^{\mathsf{rp}} \quad \forall j \in \{0,1,2\}$
        // where $\mathsf{rp}$ is the range of the PRF
        $\Gamma \leftarrow \Gamma \cup \{(i, (i_1, \ldots, i_\ell), f, \gamma^{(0)}, \gamma^{(1)}, \gamma^{(2)})\}$
        Return $(\gamma^{(0)}, \gamma^{(1)}, \gamma^{(2)})$
```

```
Algorithm $R_Y((i_1, \ldots, i_\ell, f)$
    If $((i_1, \ldots, i_\ell), f, y) \in Y$
        Return $y$
    Else
        $y \leftarrow_\$ \{0,1\}^{\mathsf{rp}}$
        $Y \leftarrow Y \cup \{((i_1, \ldots, i_\ell), f, y)\}$
        Return $y$
```

Figure 15 Hybrid $5_{j,1}$.

Note that the hardwired ciphertext is only output for query j, the outputs for the other queries are exactly equal to those in the previous hybrid. Now, for query j, the ciphertext is hardwired and output is set to be equal to what was output in the previous hybrid. It follows that the output of FE decryption remains exactly the same as in the previous hybrid. Thus, by security of FE, we have that the two hybrids are indistinguishable.

In more detail, we have a series of subhybrids, one for each $i \notin I$, where in Hybrid $5_{j,1}, i$, the change above is made to the ciphertext associated with the FE instance for party i. Let Hybrid $5_{j,1}, 0$ denote Hybrid 4 and let Hybrid $5_{j,1}, |n \setminus I|$ denote Hybrid $5_{j,1}$. We now give the formal reduction to FE semantic security for distinguishing Hybrid $5_{j,1}, i-1$ and Hybrid $5_{j,1}, i$ (for ease of notation, we assume that the indices i are consecutive). The simulator $\mathcal{B}$ is defined as follows. First $\mathcal{B}$ receives the public parameters PP from the FE challenger. Then it receives $(1^\kappa, I, (\mathsf{PP}_j)_{j \in I})$ from the hybrid distinguisher $\mathcal{A}$. Next it runs

Step 2 to Step 11 on the left hand side of Figure 15 and passes $(\mathsf{PP}_j)_{j \notin I}$ (see Step 11) to $\mathcal{A}$. It handles encryption and key generation queries as follows. On an encryption query $(i', \mathbf{x}_0, \mathbf{x}_1)$ with $i' \neq i$, the query is handled the same as $\mathcal{O}_{\mathsf{enc}}$ in Figure 15. On an encryption query $(i, \mathbf{x}_0, \mathbf{x}_1)$, a tag $T \leftarrow_{\$} \{0,1\}^\kappa$ is sampled, then $r \leftarrow \mathsf{PRF.Eval}(K_i^{\mathsf{punc}}, j \parallel \mathsf{EK}_{i,j} \parallel T)$, $\psi \leftarrow \mathsf{MIFE.Enc}(\mathsf{EK}_{i,j}, \mathbf{x}_0; r)$, $\mathsf{Trap}_0 \leftarrow (\mathsf{mode} := R, \perp, \perp, \perp, \perp)$ and $\mathsf{Trap}_1 \leftarrow (\mathsf{mode} := \mathsf{Trap}, \psi, j, \mathbf{x}_0, \mathbf{x}_1)$ are computed and $\mathcal{B}$ makes a call to the FE encryption oracle with messages $(\mathbf{x}_0, T, K_i^{\mathsf{punc}}, \mathsf{Trap}_0)$ and $(\mathbf{x}_0, T, K_i^{\mathsf{punc}}, \mathsf{Trap}_1)$, and returns the returned ciphertext. Key generation queries are handled as in $\mathcal{O}_{\mathsf{kg}}$ in Figure 15 with one exception, namely the secret keys $\mathsf{SK}_{\mathsf{FE}_{i'}}$ are computed as in $\mathsf{GenKeys}''''$ except for the case $i' = i$; the secret key $\mathsf{SK}_{\mathsf{FE}_i}$ is obtained by making a call to the FE key generation oracle for function $\mathsf{ReEnc}_{i,\mathsf{ctr}}$. If the FE challenger's bit is 0, then $\mathcal{B}$ perfectly simulates Hybrid $5_{j,1}, i-1$ and if the FE challenger's bit is 1, then $\mathcal{B}$ perfectly simulates Hybrid $5_{j,1}, i$.

Hybrid $5_{\mathbf{j},2}$: In this hybrid, we use a punctured PRF to generate the MIFE CTs in the ReEnc functionality encoded in the FE function keys. The PRF for party i is punctured at prefix j so that the randomness $r_{i,j,k}$ defined above, for $i \in [n]$, $k \in [Q_c]$ cannot be generated. All MIFE ciphertexts corresponding to other function queries can be generated as before.

In more detail:

1. For $i \in [n]$, party i samples $K_{i,j}^{\mathsf{punc}} \leftarrow \mathsf{puncF}(K_i^{\mathsf{punc}}, j)$.
2. The i^{th} encryptor computes $\mathsf{FE.enc}\big(\mathsf{MSK}_{\mathsf{FE}}, (\mathbf{x}_{0,i,k}, T_{i,k}, K_{i,j}^{\mathsf{punc}}, \mathsf{Trap})\big)$ where all other fields are set as in the previous hybrid.

During FE decryption, for any query $j' \neq j$, we now obtain:

$$r_{i,j',k} \leftarrow \mathsf{puncF}(K_{i,j}^{\mathsf{punc}}, j' \parallel \mathsf{EK}_{i,j'} \parallel T_{i,k})$$

for $k \in [Q_c]$. Everything else is as in the previous hybrid. For query j, the hardwired CT is output, and the punctured PRF key is not used to generate randomness. Please see Figure 16 for the complete description.

We have by correctness of the punctured PRF that for any $j' \neq j$, all the computed $r_{i,j',k}$ are exactly equal to those computed in the previous hybrid, where the normal PRF key was used. For query j, the PRF is not used and the hardwired value is output in both hybrids. Hence, the outputs of FE decryption are equal in both hybrids. Thus, indistinguishability follows from security of FE.

In more detail, we have a series of subhybrids, one for each $i \notin I$, where in Hybrid $5_{j,2}, i$, the change above is made to the ciphertext associated with the FE instance for party i. Let Hybrid $5_{j,2}, 0$ denote Hybrid $5_{j,1}$ and let Hybrid $5_{j,2}, |n \setminus I|$ denote Hybrid $5_{j,2}$. We now give the formal reduction to FE semantic security for distinguishing Hybrid $5_{j,2}, i-1$ and Hybrid $5_{j,2}, i$ (for ease of notation, we assume that the indices i are consecutive). The simulator $\mathcal{B}$ is defined as follows. First $\mathcal{B}$ receives the public parameters PP from the FE challenger. Then it receives $(1^\kappa, I, (\mathsf{PP}_j)_{j \in I})$ from the hybrid distinguisher $\mathcal{A}$. Next it runs Step 2 to Step 12 on the left hand side of Figure 16 and passes $(\mathsf{PP}_j)_{j \notin I}$ (see Step 12) to $\mathcal{A}$. It handles encryption and key generation queries as follows. On an encryption query $(i', \mathbf{x}_0, \mathbf{x}_1)$ with $i' \neq i$, the query is handled the same as $\mathcal{O}_{\mathsf{enc}}$ in Figure 16. On an encryption query $(i, \mathbf{x}_0, \mathbf{x}_1)$, a tag $T \leftarrow_{\$} \{0,1\}^\kappa$ is sampled, then $r \leftarrow \mathsf{PRF.Eval}(K_i^{\mathsf{punc}}, j \parallel \mathsf{EK}_{i,j} \parallel T)$, $\psi \leftarrow \mathsf{MIFE.Enc}(\mathsf{EK}_{i,j}, \mathbf{x}_0; r)$ and $\mathsf{Trap} \leftarrow (\mathsf{mode} := \mathsf{Trap}, \psi, j, \mathbf{x}_0, \mathbf{x}_1)$ are computed and $\mathcal{B}$ makes a call to the FE encryption oracle with messages $(\mathbf{x}_0, T, K_i^{\mathsf{punc}}, \mathsf{Trap})$ and $(\mathbf{x}_0, T, K_{i,j}^{\mathsf{punc}}, \mathsf{Trap})$, and returns the returned ciphertext. Note that the punctured key $K_{i,j}^{\mathsf{punc}}$ is derived in Step 4 on the left hand side of Figure 16. Key generation queries are handled as in $\mathcal{O}_{\mathsf{kg}}$ in Figure 15 with one exception, namely the secret keys $\mathsf{SK}_{\mathsf{FE}_{i'}}$ are computed as in $\mathsf{GenKeys}''''$ except for the case $i' = i$; the

```
Hybrid $\mathbf{5_{j,2}}$:
    $(1^n, I, (\mathsf{PP}_i)_{i\in I}, st) \leftarrow_\$ \mathcal{A}_0(1^\kappa)$
    $b \leftarrow_\$ \{0,1\}$
    $\forall i \notin I$:
    $K_i^{\mathsf{punc}} \leftarrow_\$ \mathcal{K}_\kappa$
    $K_{i,\mathsf{j}}^{\mathsf{punc}} \leftarrow \mathsf{PRF.Punc}(K_i^{\mathsf{punc}}, j)$
    $(\mathsf{PP}_{\mathsf{FE}_i}, \mathsf{MSK}_{\mathsf{FE}_i}) \leftarrow_\$ \mathsf{FE.Setup}(1^\kappa)$
    $\Gamma \leftarrow \emptyset$
    $Y \leftarrow \emptyset$
    $Q \leftarrow \emptyset$
    $q \leftarrow 0$
    $\mathsf{mkeys} := ((\mathsf{EK}_{1,\mathsf{j}}, \ldots, \mathsf{EK}_{n,\mathsf{j}}), \mathsf{MSK}_{\mathsf{j}})$
    $\leftarrow_\$ \mathsf{MIFE.Setup}(1^\kappa)$
    $(\rho_i^{(1)})_{i\notin I} \leftarrow_\$ \widetilde{\mathsf{Sim}}_1()$
    $\mathsf{PP}_i \leftarrow \rho_i^{(1)} \quad \forall i \notin I$

    $st \leftarrow_\$ \mathcal{A}_1^{\mathcal{O}_{\mathsf{enc}}(\cdot,\cdot,\cdot),\mathcal{O}_{\mathsf{kg}}(\cdot,\cdot,\cdot)}(st)$
    $b' \leftarrow_\$ \mathcal{A}_2^{\mathcal{O}_{\mathsf{enc}}(\cdot,\cdot,\cdot),\mathcal{O}_{\mathsf{kg}}(\cdot,\cdot,\cdot)}((\mathsf{PP}_i)_{i\notin I}, st)$
    Return $(b = b')$

Algorithm $\mathsf{GenKeys}_f''''((r_k', \mathsf{MSK}_{\mathsf{FE}_k})_{k\in[\ell]}, \mathsf{ctr}, \mathsf{mkeys})$
    If $\mathsf{ctr} = \mathsf{j}$
        $((\mathsf{EK}_1, \ldots, \mathsf{EK}_\ell), \mathsf{MSK}) \leftarrow \mathsf{mkeys}$
    Else
        $((\mathsf{EK}_1, \ldots, \mathsf{EK}_\ell), \mathsf{MSK}) \leftarrow \mathsf{MIFE.Setup}(1^\kappa)$
    $\mathsf{SK}_f \leftarrow \mathsf{MIFE.KeyGen}(\mathsf{MSK}, f)$
    $\forall i \in [\ell]$, do:
    $\mathsf{SK}_{\mathsf{FE}_i} \leftarrow \mathsf{FE.KeyGen}(\mathsf{MSK}_{\mathsf{FE}_i}, \mathsf{ReEnc}_{\mathsf{EK}_i,\mathsf{ctr}}; r_i')$
    Return $(\mathsf{SK}_f, \mathsf{SK}_{\mathsf{FE}_1}, \ldots, \mathsf{SK}_{\mathsf{FE}_\ell})$
```

```
Oracle $\mathcal{O}_{\mathsf{enc}}(i, \mathbf{x}_0, \mathbf{x}_1)$
    $T \leftarrow_\$ \{0,1\}^\kappa$
    $r \leftarrow \mathsf{PRF.Eval}(K_i^{\mathsf{punc}}, \mathsf{j} \parallel \mathsf{EK}_{i,\mathsf{j}} \parallel T)$
    $\psi \leftarrow \mathsf{MIFE.Enc}(\mathsf{EK}_{i,\mathsf{j}}, \mathbf{x}_0; r)$
    $\mathsf{Trap} \leftarrow (\mathsf{mode} := \mathsf{Trap}, \mathsf{CT} := \psi,$
    $\mathsf{index} := \mathsf{j}, \mathsf{val}_0 := \mathbf{x}_0, \mathsf{val}_1 := \mathbf{x}_1)$
    Return $\mathsf{FE.Enc}(\mathsf{MSK}_{\mathsf{FE}_i}, (\mathbf{x}_0, T, K_{i,\mathsf{j}}^{\mathsf{punc}}, \mathsf{Trap}))$
```

```
Oracle $\mathcal{O}_{\mathsf{kg}}(i, (i_1, \ldots, i_\ell), f)$
    $J \leftarrow I \cap \{i_1, \ldots, i_\ell\}$; $\bar{J} \leftarrow \{i_1, \ldots, i_\ell\} \setminus J$
    $(\alpha_j, r_j, K_j) \leftarrow \widetilde{\mathsf{Extract}}(\rho_j^{(1)}) \quad \forall j \in J$
    $y \leftarrow_\$ R_Y((i_1, \ldots, i_\ell), f)$
    If $((i_1, \ldots, i_\ell), f, \bar{J}') \notin Q$
        $\bar{J}' \leftarrow \emptyset$
    $\bar{J}' \leftarrow \bar{J}' \cup \{i\}$
    $Q \leftarrow Q \cup ((i_1, \ldots, i_\ell), f, \bar{J}')$
    $(\rho_j^{(2)})_{j\in\bar{J}} \leftarrow_\$ \widetilde{\mathsf{Sim}}_2((\alpha_j, r_j, K_j)_{j\in J}, y)$
    $\forall j \in \bar{J}$:
    $(\gamma_j^{(0)}, \gamma_j^{(1)}, \gamma_j^{(2)}) \leftarrow_\$ R_\Gamma(j, (i_1, \ldots, i_\ell), f)$
    $\forall m \in \{0,1,2\}, j \in J$,
    $\gamma_j^{(m)} \leftarrow \mathsf{PRF.Eval}(K_j, m \parallel (\mathsf{PP}_{i_k})_{k\in[\ell]} \parallel f)$
    If $\bar{J}' = \bar{J}$ // if fully queried
        $q \leftarrow q + 1$
        $S_1 \leftarrow_\$ \bigoplus_{j\in\bar{J}, j\neq i} \gamma_j^{(0)}$
        $S_2 \leftarrow_\$ \bigoplus_{j\in J} \gamma_j^{(0)}$
        $s \leftarrow y \oplus S_1 \oplus S_2 \oplus$
        $\mathsf{GenKeys}''''(\{(\gamma_{i_k}^{(2)}, \mathsf{MSK}_{\mathsf{FE}_{i_k}})\}_{k\in[\ell]}, q, \mathsf{mkeys})$
    Else
        $s \leftarrow \gamma_i^{(0)}$
    Return $(\rho_i^{(2)}, s)$
```

```
Algorithm $R_\Gamma(i, (i_1, \ldots, i_\ell, f)$
    If $(i, (i_1, \ldots, i_\ell), f, \gamma^{(0)}, \gamma^{(1)}, \gamma^{(2)}) \in \Gamma$
        Return $(\gamma^{(0)}, \gamma^{(1)}, \gamma^{(2)})$
    Else
        $\gamma^{(j)} \leftarrow_\$ \{0,1\}^{\mathsf{rp}} \quad \forall j \in \{0,1,2\}$
        // where rp is the range of the PRF
        $\Gamma \leftarrow \Gamma \cup \{(i, (i_1, \ldots, i_\ell), f, \gamma^{(0)}, \gamma^{(1)}, \gamma^{(2)})\}$
        Return $(\gamma^{(0)}, \gamma^{(1)}, \gamma^{(2)})$
```

```
Algorithm $R_Y((i_1, \ldots, i_\ell, f)$
    If $((i_1, \ldots, i_\ell), f, y) \in Y$
        Return $y$
    Else
        $y \leftarrow_\$ \{0,1\}^{\mathsf{rp}}$
        $Y \leftarrow Y \cup \{((i_1, \ldots, i_\ell), f, y)\}$
        Return $y$
```

Figure 16 Hybrid $5_{j,2}$.

secret key $\mathsf{SK}_{\mathsf{FE}_i}$ is obtained by making a call to the FE key generation oracle for function $\mathsf{ReEnc}_{i,\mathsf{ctr}}$. If the FE challenger's bit is 0, then $\mathcal{B}$ perfectly simulates Hybrid $5_{j,2}, i-1$ and if the FE challenger's bit is 1, then $\mathcal{B}$ perfectly simulates Hybrid $5_{j,2}, i$.

Hybrid $\mathbf{5_{j,3}}$: In this hybrid, we switch the randomness used in the hardwired MIFE CT to be true randomness. That is, $r_{i,j,k}$ is sampled uniformly at random for $i \in [n]$, $k \in [Q_c]$. We have by the security of the punctured PRF that given the punctured key, the PRF evaluations at the punctured points are indistinguishable from random. Hence, indistinguishability follows from security of punctured PRF.

Please see Figure 17 for the detailed description.

```
**Hybrid $5_{j,3}$:**
    $(1^n, I, (\mathsf{PP}_i)_{i \in I}, st) \leftarrow\$ \mathcal{A}_0(1^\kappa)$
    $b \leftarrow\$ \{0,1\}$
    $\forall i \notin I$:
    $K_i^{\mathsf{punc}} \leftarrow\$ \mathcal{K}_\kappa$
    $K_{i,\mathrm{j}}^{\mathsf{punc}} \leftarrow \mathsf{PRF.Punc}(K_i^{\mathsf{punc}}, j)$
    $(\mathsf{PP}_{\mathsf{FE}_i}, \mathsf{MSK}_{\mathsf{FE}_i}) \leftarrow\$ \mathsf{FE.Setup}(1^\kappa)$
    $\Gamma \leftarrow \emptyset$
    $Y \leftarrow \emptyset$
    $Q \leftarrow \emptyset$
    $q \leftarrow 0$
    $\mathsf{mkeys} := ((\mathsf{EK}_{1,\mathrm{j}}, \ldots, \mathsf{EK}_{n,\mathrm{j}}), \mathsf{MSK}_\mathrm{j}) \leftarrow\$ \mathsf{MIFE.Setup}(1^\kappa)$
    $(\rho_i^{(1)})_{i \notin I} \leftarrow\$ \widetilde{\mathsf{Sim}}_1()$
    $\mathsf{PP}_i \leftarrow \rho_i^{(1)} \quad \forall i \notin I$

    $st \leftarrow\$ \mathcal{A}_1^{\mathcal{O}_{\mathsf{enc}}(\cdot,\cdot,\cdot), \mathcal{O}_{\mathsf{kg}}(\cdot,\cdot,\cdot)}(st)$
    $b' \leftarrow\$ \mathcal{A}_2^{\mathcal{O}_{\mathsf{enc}}(\cdot,\cdot,\cdot), \mathcal{O}_{\mathsf{kg}}(\cdot,\cdot,\cdot)}((\mathsf{PP}_i)_{i \notin I}, st)$
    Return $(b = b')$

**Algorithm** $\mathsf{GenKeys}_f''''((r'_k, \mathsf{MSK}_{\mathsf{FE}_k})_{k \in [\ell]}, \mathsf{ctr}, \mathsf{mkeys})$
    If $\mathsf{ctr} = \mathrm{j}$
        $((\mathsf{EK}_1, \ldots, \mathsf{EK}_\ell), \mathsf{MSK}) \leftarrow \mathsf{mkeys}$
    Else
        $((\mathsf{EK}_1, \ldots, \mathsf{EK}_\ell), \mathsf{MSK}) \leftarrow \mathsf{MIFE.Setup}(1^\kappa)$
    $\mathsf{SK}_f \leftarrow \mathsf{MIFE.KeyGen}(\mathsf{MSK}, f)$
    $\forall i \in [\ell]$, do:
    $\mathsf{SK}_{\mathsf{FE}_i} \leftarrow \mathsf{FE.KeyGen}(\mathsf{MSK}_{\mathsf{FE}_i}, \mathsf{ReEnc}_{\mathsf{EK}_i, \mathsf{ctr}}; r'_i)$
    Return $(\mathsf{SK}_f, \mathsf{SK}_{\mathsf{FE}_1}, \ldots, \mathsf{SK}_{\mathsf{FE}_\ell})$
```

```
**Oracle** $\mathcal{O}_{\mathsf{enc}}(i, \mathbf{x}_0, \mathbf{x}_1)$
    $T \leftarrow\$ \{0,1\}^\kappa$
    $r \leftarrow\$ \{0,1\}^{rp}$
        // where rp is the range of the PRF
    $\psi \leftarrow \mathsf{MIFE.Enc}(\mathsf{EK}_{i,\mathrm{j}}, \mathbf{x}_0; r)$
    $\mathsf{Trap} \leftarrow (\mathsf{mode} := \mathsf{Trap}, \mathsf{CT} := \psi, \mathsf{index} := \mathrm{j},$
    $\mathsf{val}_0 := \mathbf{x}_0, \mathsf{val}_1 := \mathbf{x}_1)$
    Return $\mathsf{FE.Enc}(\mathsf{MSK}_{\mathsf{FE}_i}, (\mathbf{x}_0, T, K_{i,\mathrm{j}}^{\mathsf{punc}}, \mathsf{Trap}))$
```

```
**Oracle** $\mathcal{O}_{\mathsf{kg}}(i, (i_1, \ldots, i_\ell), f)$
    $J \leftarrow I \cap \{i_1, \ldots, i_\ell\}$; $\bar{J} \leftarrow \{i_1, \ldots, i_\ell\} \setminus J$
    $(\alpha_j, r_j, K_j) \leftarrow \widetilde{\mathsf{Extract}}(\rho_j^{(1)}) \quad \forall j \in J$
    $y \leftarrow\$ R_Y((i_1, \ldots, i_\ell), f)$
    If $((i_1, \ldots, i_\ell), f, \bar{J}') \notin Q$
        $\bar{J}' \leftarrow \emptyset$
    $\bar{J}' \leftarrow \bar{J}' \cup \{i\}$
    $Q \leftarrow Q \cup ((i_1, \ldots, i_\ell), f, \bar{J}')$
    $(\rho_j^{(2)})_{j \in \bar{J}} \leftarrow\$ \widetilde{\mathsf{Sim}}_2((\alpha_j, r_j, K_j)_{j \in J}, y)$
    $\forall j \in \bar{J}$:
    $(\gamma_j^{(0)}, \gamma_j^{(1)}, \gamma_j^{(2)}) \leftarrow\$ R_\Gamma(j, (i_1, \ldots, i_\ell), f)$
    $\forall m \in \{0,1,2\}, j \in J,$
    $\gamma_j^{(m)} \leftarrow \mathsf{PRF.Eval}(K_j, m \parallel (\mathsf{PP}_{i_k})_{k \in [\ell]} \parallel f)$
    If $\bar{J}' = \bar{J}$ // if *fully queried*
        $q \leftarrow q + 1$
        $S_1 \leftarrow\$ \bigoplus_{j \in \bar{J}, j \neq i} \gamma_j^{(0)}$
        $S_2 \leftarrow\$ \bigoplus_{j \in J} \gamma_j^{(0)}$
        $s \leftarrow y \oplus S_1 \oplus S_2 \oplus$
        $\mathsf{GenKeys}''''(\{(\gamma_{i_k}^{(2)}, \mathsf{MSK}_{\mathsf{FE}_{i_k}})\}_{k \in [\ell]}, q, \mathsf{mkeys})$
    Else
        $s \leftarrow \gamma_i^{(0)}$
    Return $(\rho_i^{(2)}, s)$
```

```
**Algorithm** $R_\Gamma(i, (i_1, \ldots, i_\ell, f)$
    If $(i, (i_1, \ldots, i_\ell), f, \gamma^{(0)}, \gamma^{(1)}, \gamma^{(2)}) \in \Gamma$
        Return $(\gamma^{(0)}, \gamma^{(1)}, \gamma^{(2)})$
    Else
        $\gamma^{(j)} \leftarrow\$ \{0,1\}^{\mathsf{rp}} \quad \forall j \in \{0,1,2\}$
        // where rp is the range of the PRF
        $\Gamma \leftarrow \Gamma \cup \{(i, (i_1, \ldots, i_\ell), f, \gamma^{(0)}, \gamma^{(1)}, \gamma^{(2)})\}$
        Return $(\gamma^{(0)}, \gamma^{(1)}, \gamma^{(2)})$
```

```
**Algorithm** $R_Y((i_1, \ldots, i_\ell, f)$
    If $((i_1, \ldots, i_\ell), f, y) \in Y$
        Return $y$
    Else
        $y \leftarrow\$ \{0,1\}^{\mathsf{rp}}$
        $Y \leftarrow Y \cup \{((i_1, \ldots, i_\ell), f, y)\}$
        Return $y$
```

Figure 17 Hybrid $5_{j,3}$.

In more detail, we have a series of subhybrids, one for each $i \notin I$, where in Hybrid $5_{j,3}, i$, the change above is made to the function key associated with the FE instance for party i. Let Hybrid $5_{j,3}, 0$ denote Hybrid $5_{j,2}$ and let Hybrid $5_{j,3}, |n \setminus I|$ denote Hybrid $5_{j,3}$. We now give the formal reduction to PRF security for distinguishing Hybrid $5_{j,3}, i-1$ and Hybrid $5_{j,3}, i$ (for ease of notation, we assume that the indices i are consecutive). The simulator $\mathcal{B}$ is defined as follows. First $\mathcal{B}$ receives $(1^\kappa, I, (\mathsf{PP}_j)_{j \in I})$ from the hybrid distinguisher $\mathcal{A}$. Next it runs

- For all $i' \notin I$, $i' \neq i$:
  - $K_{i'}^{\mathsf{punc}} \leftarrow\$ \mathcal{K}_\kappa$
  - $K_{i',j}^{\mathsf{punc}} \leftarrow \mathsf{PRF.Punc}(K_{i'}^{\mathsf{punc}}, j)$

Then it sends a challenge prefix j to the PRF challenger and receives a punctured key; call this $K_{i,j}^{\mathsf{punc}}$. Next $\mathcal{B}$ runs Step 5 to Step 12 on the left hand side of Figure 17 and passes $(\mathsf{PP}_j)_{j \notin I}$ (see Step 12) to $\mathcal{A}$. It handles encryption and key generation queries as follows. Key generation queries are handled as in $\mathcal{O}_{\mathsf{kg}}$ in Figure 17. On an encryption query $(i', \mathbf{x}_0, \mathbf{x}_1)$ with $i' \neq i$, the query is handled the same as $\mathcal{O}_{\mathsf{enc}}$ in Figure 17. On an encryption query $(i, \mathbf{x}_0, \mathbf{x}_1)$, the query is handled the same as $\mathcal{O}_{\mathsf{enc}}$ in Figure 17 except Step 2 of $\mathcal{O}_{\mathsf{enc}}$ where r is computed. To obtain r, the PRF evaluation oracle is queried at the point $j \parallel \mathsf{EK}_{i,j} \parallel T$ where T is derived in Step 1 of $\mathcal{O}_{\mathsf{enc}}$. Now if the PRF challenger's bit is 0, the string r will be computed using the PRF and $\mathcal{B}$ perfectly simulates $5_{j,3}, i-1$. On the other hand, if the PRF challenger's bit is 1, the string r will be uniformly random because the queried evaluation point begins with the challenge prefix j (i.e. the PRF is punctured at that point) and so $\mathcal{B}$ perfectly simulates Hybrid $5_{j,3}, i$.

Hybrid $\mathbf{5_{j,4}}$: In this Hybrid, the protocol $\mathsf{GenKeys}''''$ is modified further so that for key j, $\mathsf{MIFE.Setup}$ or $\mathsf{MIFE.KeyGen}$ are not invoked. Rather, the output of the $\mathsf{MIFE.KeyGen}$ algorithm is hardwired and output for key j. The hardwired value is exactly the same as in the previous hybrid, hence indistinguishability holds by security of MPC. In more detail, the MPC simulator receives identical inputs in both hybrids and hence produces indistinguishable outputs in the two hybrids.

Hybrid $\mathbf{5_{j,5}}$: In this hybrid, we switch the hardwired MIFE CTs within the FE CTs to use bit $b = 1$. Indistinguishability follows from MIFE security. In more detail, we fix query j. For party $i \in [n]$ and ciphertext query $k \in [Q_c]$, we construct a reduction which plays against the MIFE challenger as below. The reduction computes everything as in the previous hybrid except that the hardwired ciphertexts for the j^{th} copy of MIFE, which it receives from the MIFE challenger as below:

1. The reduction requests for MIFE key corresponding to function f_j which it receives. It hardwires this into functionality $\mathsf{GenKeys}$.
2. For $i \in [n]$, $k \in [Q_c]$, the challenge ciphertexts for party i are set as $(\mathbf{x}_{0,i,k}, T_{i,k}, K_{i,j}^{\mathsf{punc}})$ and $(\mathbf{x}_{1,i,k}, T_{i,k}, K_{i,j}^{\mathsf{punc}})$.
3. The reduction receives $\psi_{i,j,k}$ for $i \in [n]$ and $k \in [Q_c]$. It hardwires these into the FE ciphertexts as discussed above.

It is evident that if $b = 0$ then we are in Hybrid $\mathbf{5_{j,4}}$ and if $b = 1$ we are in Hybrid $\mathbf{5_{j,5}}$. Thus, an adversary that distinguishes between these two hybrids can be used to break the security of the j^{th} MIFE scheme.

Now that the bit b has been switched for query j, we roll back our changes. Arguments of indistinguishability are analogous to the above and are skipped.

Hybrid $\mathbf{5_{j,6}}$: Change $\mathsf{GenKeys}''$ to invoke $\mathsf{MIFE.Setup}$ and $\mathsf{MIFE.KeyGen}$ as before.

Hybrid $\mathbf{5_{j,7}}$: Switch randomness in the hardwired CT back to PRF randomness.

Hybrid $\mathbf{5_{j,8}}$: Switch punctured PRF key back to normal PRF key.

Hybrid $\mathbf{5_{j,9}}$: Increment $\mathsf{Trap.index}$ by 1. At this point, for key j, we have $\mathsf{ctr_j} < \mathsf{Trap_{index}}$, hence by the design of the ReEnc algorithm, we have that the bit $b = 1$ is used for MIFE encryption. This is indistinguishable from the previous hybrid by security of FE because decryption values are exactly the same in both the hybrids.

Hybrid $\mathbf{5_{j+1,1}}$: This Hybrid is analogous to Hybrid $\mathbf{5_{j,1}}$. Indistinguishability follows by security of FE as discussed above. Finally, in Hybrid $\mathbf{5_{Q_k,9}}$, all the keys are outputting MIFE CTs corresponding to $b = 1$.

Hybrid 6: In this hybrid, use message corresponding to $b = 1$ and $\mathsf{mode} = R$. Again, indistinguishability follows by security of FE since the outputs are the same.

Hybrid 7: Undo the changes made in Hybrid 4, namely the algorithm $\mathsf{GenKeys}''$ invoked for query index j for all $j \in [Q_k]$ is modified to invoke ReEnc with parameter $\mathsf{ctr} = \perp$. Indistinguishability follows analogously to the transition between Hybrid 3 and Hybrid 4.

Hybrid 8: Undo the changes made in Hybrid 3. Specifically, we generate y (the target output passed to $\widetilde{\mathsf{Sim2}}$) and s (the masking value component of the partial decryption key) the same as in Hybrid 2 for each query $((i_1, \ldots, i_\ell), f)$. Indistinguishability follows analogously to the transition between Hybrid 2 and Hybrid 3.

Hybrid 9: Undo the changes made in Hybrid 2. More precisely, we replace the uniformly random strings used in the function $\mathsf{GenKeys}$ for the honest users i with the outputs of the PRF. Indistinguishability follows analogously to the transition between Hybrid 1 and Hybrid 2.

Hybrid 10: Undo the changes made in Hybrid 1, that is, we generate the first-round protocol messages $\rho_i^{(1)}$ (in the public parameters) and second-round protocol messages $\rho_i^{(2)}$ (in the partial decryption keys) for MPC as in the real system for each uncorrupted user $i \notin I$. Indistinguishability follows analogously to the transition between Hybrid 0 and Hybrid 1.

Hybrid 10 is the real world with bit $b = 1$. ◀

E Proof of Security for Ad Hoc MIFE for Inner Products

In this section, we provide the proof of security for our ad hoc MIFE for inner products.

▶ **Theorem 10.** *If the* MIFE *constructed by [2] is a* xxx*-IND-secure MIFE scheme and* MPC *is an* xxx*-SIM-secure 2-round MPC protocol, then our construction is* sel*-IND-secure.*

Proof. The proof follows easily from the proof of theorem 5. For simplicity, we describe the proof for the case of a single key query. The case of multiple queries is handled exactly as in the proof of theorem 5. In more detail, we define:

Hybrid 0: This is the real game in which on every encryption query $(i, \mathbf{x}_0, \mathbf{x}_1)$, $\mathbf{x}_0$ is encrypted.

Hybrid 1: Exactly as in the proof of theorem 5, the MPC transcript in this hybrid is simulated. Indistinguishability follows as in the proof of theorem 5.

Hybrid 2: In this hybrid, we switch the bit in the MIFE ciphertexts to 1. Indistinguishability follows via a reduction, in which the MIFE function key and ciphertexts are obtained from the MIFE challenger. The MIFE function key is input to the MPC simulator and the MPC transcript and MIFE ciphertexts are returned to the adversary.

To support multiple keys, we proceed as in the proof of theorem 5 and change the bit used in the MIFE encryption from 0 to 1 key by key. ◀