Dynamic Decentralized Functional Encryption

Jérémy Chotard[1,2,3], Edouard Dufour-Sans[2,3,4], Romain Gay[5], Duong Hieu Phan[1], and David Pointcheval[2,3]

[1] XLIM, University of Limoges, CNRS, Limoges, France
[2] DIENS, École normale supérieure, CNRS, PSL University, Paris, France
[3] INRIA, Paris, France
[4] Carnegie Mellon University, Pittsburgh, USA
[5] Cornell Tech, New York, USA

Abstract. We introduce Dynamic Decentralized Functional Encryption (DDFE), a generalization of Functional Encryption which allows multiple users to join the system dynamically, without relying on a trusted third party or on expensive and interactive Multi-Party Computation protocols. This notion subsumes existing multi-user extensions of Functional Encryption, such as Multi-Input, Multi-Client, and Ad Hoc Multi-Input Functional Encryption.
We define and construct schemes for various functionalities which serve as building-blocks for latter primitives and may be useful in their own right, such as a scheme for dynamically computing sums in any Abelian group. These constructions build upon simple primitives in a modular way, and have instantiations from well-studied assumptions, such as DDH or LWE.
Our constructions culminate in an Inner-Product scheme for computing weighted sums on aggregated encrypted data, from standard assumptions in prime-order groups in the Random Oracle Model.

Keywords. Dynamic, Decentralized, Functional Encryption, Inner Product.

1 Introduction

At TCC'11, Boneh, Sahai, and Waters [11] formalized Functional Encryption (FE), a new paradigm of Public-Key Encryption that allows the owner of the secret key to generate restricted keys, enabling third parties to recover *function evaluations* of the plaintext from a ciphertext. The formalization of FE gave many researchers a common framework in which to consider their schemes: the nuances between Identity-Based Encryption (IBE), Hierarchical IBE, Fuzzy IBE, and different forms of Attribute-Based Encryption (ABE) [9, 10, 27, 32] could now be captured simply by specifying which functionality the scheme aims to implement. The set of algorithms to be implemented and the indistinguishibility game in which to prove security were now standard.

But for all its successes, Functional Encryption has two, somewhat related, important limitations: (1) In many contexts, FE encourages centralization. In his 2015 position paper *The Moral Character of Cryptographic Work*, Rogaway pointed out that a switch from Public-Key Encryption to Identity-Based Encryption would represent "a radical change in the trust model", as the authority with knowledge of the master secret key would have the ability to fully recover every message encrypted under its public key, even though those messages would be intended for a variety of parties. This criticism can be extended to many other functionalities of Functional Encryption. (2) The kind of controlled computation enabled by Functional Encryption does not extend to computations involving data from multiple parties. This is limiting because a significant component of the public's privacy concerns today is related to data being made available to a third-party for the advertised purpose of retrieving some form of intelligence of the public's needs, from the computation of simple statistics to the training of advanced machine learning models. This means FE is not an appropriate framework for addressing this pressing issue.

1.1 Our Contributions

1. First, we fill the gap left by the definition of FE by introducing a new primitive we term Dynamic Decentralized Functional Encryption (DDFE). DDFE allows aggregating data coming from different parties, does not require a trusted party with a master secret key, and accounts for participants wanting to join at various stages during the lifetime of a system. Previous extensions of FE, which we review in more detail in Section 1.2, either failed to address the concerns we raised above, or partially forwent the generality that made the success of FE as a framework for describing cryptographic schemes. We give a formal definition of DDFE as well as a security definition.
2. We define All-or-Nothing Encapsulation (AoNE), a functionality of DDFE which we found to be a critical building-block when constructing useful DDFE schemes later in this work. AoNE allows a participant to send its data to be aggregated with other data coming from a group of participants agreeing on a label ℓ. Only if all those participants choose to send data for aggregation with the same group under the same label will the data of all participants be revealed, otherwise, nothing is revealed. We provide two constructions of AoNE. The first one is generic from any IBE, but has individual ciphertexts that grow linearly in the number of participants in an aggregation, which is not ideal. The second construction is specific and achieves constant size ciphertexts. It relies on bilinear maps, and we prove its security under the DBDH assumption in the Random Oracle Model (ROM).
3. We define and provide a construction of DSum, a functionality of DDFE which is both interesting in its own right and a useful building-block for other constructions. DSum operates over any Abelian group and allows multiple parties to send an element from that group for aggregation with a set of participants agreeing on a label ℓ. Once every participant has sent data for aggregation with that set and that label, the sum (or rather the repeated group operation) of the data is revealed. We provide a generic construction of DSum from Non-Interactive Key Exchange (NIKE), AoNE DDFE and Pseudo-Random Functions (PRF).

4. We define and provide a construction of Inner-Product DDFE (IP-DDFE), which allows for more complex patterns of aggregation than DSum. In IP-DDFE, participants can contribute to the generation of functional decryption keys that enable individuals to compute weighted sums of plaintext data, with the weights being encoded in the key. Our construction relies on AoNE, DSum, Single-Input Inner-Product Functional Encryption, and PRFs, and we prove that it is selectively secure under the DDH assumption in the ROM.

1.2 Related Work

Fully Homomorphic Encryption (FHE) [23] is a commonly cited as a cryptographic solution to issues involving computations on encrypted data at large. We stress here that FHE shines when computation delegation is intended. That is, it is useful when a client, owning some data it wishes to protect the confidentiality of, wants a server to perform computations on their data without seeing the data. This scenario arises when the computation depends on parameters known only to the server (as in the case of Information Retrieval), or when the client wants to leverage the computational power of the server.
In the scenarii we are concerned with, however, the server directly learns something about the aggregated data, without interacting with them. This stands in contrast with FHE, where the parties need to engage in extra rounds of interaction to perform a joint decryption of the encrypted data.
FE enables the server to recover information as controlled by the client through key delegation, while FHE does not limit the types of computations the server can perform, but prevents the server from accessing any data. Given these advantages, we naturally focus on extending the line of works involving FE.
Note that FHE was also initially defined for a single data owner, and was later extended to multiple users under the name Multi-Key FHE [31].

Private Stream Aggregation (PSA). This notion, initially termed Privacy-Preserving Aggregation of Time-Series Data, is an early primitive for non-interactive aggregation of multi-party data introduced by Shi *et al.* [34]. Unlike our DDFE schemes, PSA, under its standard definitions, relies on a trusted third-party distributing the participant's secret keys, cannot accommodate new participants, and does not allow the participants to choose which functions can be computed by whom via functional decryption key derivation. Most PSA schemes in the literature focus on computing (non-weighted) sums of the participants' data [8, 14, 28]. Note that Private Stream Aggregation usually relies on a Differential Privacy component as an added privacy protection, while we leave the addition of a Differential Privacy layer in DDFE for future work.

Multi-Authority Functional Encryption (MAFE) was introduced by Chandran *et al.* [15]. Like DDFE, it is a strongly decentralized variant of Functional

Encryption. It allows for encrypting messages for sets of authorities along with an access policy. These authorities can then generate keys for individual identities. Armed with a single ciphertext and a set of functional decryption keys from the appropriate authorities, the decrypter can recover a function of the plaintext that is specified by the access policy on the identities for which the functional keys were computed. Unlike DDFE, MAFE does not account for the possibility of multiple ciphertexts being decrypted together, and having their data interact with one another.

Multi-Client Functional Encryption (MCFE) was defined in [25, 26] along with Multi-Input Functional Encryption (MIFE), and also enables computing functions of multiple parties' data in the presence of a trusted third-party distributing both the parties' secret keys and functional decryption keys. That is, both MIFE and MCFE extend Functional Encryption to a setting where the input is spread across different sources. Each source can encrypt its data independently, and the ciphertexts can then be aggregated and decrypted with functional decryption keys. Generation of the latter still requires a trusted authority, which owns a so-called master secret key: a single point of failure for the cryptosystem.

As opposed to MIFE, the encryption algorithm of an MCFE takes an additional input, referred to as a label, which enforces a finer-grained control on access to the encrypted data. Unlike in MIFE, where individual ciphertexts can be arbitrarily combined, in MCFE, only ciphertexts generated for the same label can be used together to decrypt. This limits how much information is revealed by each functional decryption key, thereby strengthening security. Typically, labels are used as timestamps. In this context, a functional decryption key can only compute, say, statistics on aggregated data *for the same time frame*.

Any MCFE for a given functionality directly implies an MIFE for the same functionality, by simply using a fixed label for all encryptions[6]. Reciprocally, an MIFE for general functions would directly imply an MCFE for general functions, since the label can be part of the plaintext, and the function can check that every slot used the same label. However, this is not true for the case of smaller classes of functions for which there are practical schemes, such as Inner-Products.

The first construction of MIFE for inner products was given in [5], from standard assumptions in pairing groups. This was later improved by [4], which gave a generic construction from any single-input FE for inner products. The first construction of MCFE from standard assumptions was given by Chotard *et al.* [17] for computing inner products, although the security they achieved admits several limitations compared to the standard MCFE security definition.

Decentralized Multi-Client Functional Encryption (DMCFE). Chotard *et al.* [17] also defined a new variant of MCFE, called Decentralized MCFE (DMCFE), for which they gave Inner-Product instantiations from pairings. The

[6] Note that this was not true for MCFE as originally defined in [25], as that definition had strictly increasing timestamps for labels. But followup works on MCFE have usually allowed any bitstring to be used as a label, opening the primitive to the possibility of repetitions

DMCFE variant did away with the trusted third-party, as it enabled participants to choose their own secret keys and generate functional decryption keys non-interactively. However, it still had an interactive setup, with no easy way of adding new participants, and it suffered from the same security caveats as the MCFE it was a variant of.

In a follow-up work, [30] provided a construction in the standard model from the LWE assumption, which still suffers from the same security restrictions as [17]. The works [1, 2] improved the security guarantees obtained, the former using the DDH assumption in the ROM, the latter using a generic construction from any single-input FE for inner products. Both schemes however have individual ciphertexts of size proportional to the total number of users. Thus, we use different techniques to obtain the desirable security notion without having asymptotically large ciphertexts.

Ad Hoc Multi-Input Functional Encryption. In [6], the authors define the notion of Ad Hoc Multi-Input Functional Encryption, where users can join the system on-the-fly, and functional decryption keys can be generated in a decentralized way, by each client, without interaction. They give a feasibility result for all functions, and a practical construction for inner products.

The definition of DDFE we put forth is more general than [6]. For instance, in our definition, the algorithm that generates functional decryption key does not necessarily require a specified group of users: schemes with potentially more flexibility than Ad Hoc MIFE can be captured by our definition.

Moreover, their scheme for inner product cannot handle labels, which implies that ciphertexts computed by each client individually can be mix and matched arbitrarily. As explained above, this implies that each functional decryption key reveals large amounts of information on the encrypted values, and renders the security vacuous whenever sufficiently many keys are issued. Labels help mitigate this leakage by enforcing a better granularity on the way the encrypted data is accessed.

Besides, the security model of [6] does not explicitly address the information that can be leaked when decrypting partial ciphertexts, that is, ciphertexts coming from an incomplete group of users. Preventing the adversary from recovering information on partial ciphertexts is made more challenging in our construction, which handles labels.

1.3 Outline

We first provide a definition of DDFE in Section 2, along with a security definition and functionalities of interest. In Section 3, we recall some useful preliminaries and definitions. We then showcase our constructions: a generic construction of AoNE is presented in Section 4, while a specific instantiation is studied in Section 5. We use it modularly in Section 6 to construct a DSum scheme. In Section 7, we capitalize on both those primitives to construct a DDFE scheme for the Inner-Product functionality.

2 Definitions and Security Models

In this section, we provide the formal definition of our new primitive of *Dynamic Decentralized Functional Encryption* (DDFE), together with several security models. Then, we list a few instantiations with some concrete functionalities.

2.1 Notations

In the following, $[n]$ will denote the set of integers $\{1, \ldots, n\}$. For any set $\mathcal{A}$, $\mathcal{L}(\mathcal{A})$ will denote the set of finite lists of elements of $\mathcal{A}$, while $\mathcal{S}(\mathcal{A})$ will denote the set of finite subsets of $\mathcal{A}$. Unlike sets, lists are ordered and may contain repeated elements.

2.2 Dynamic Decentralized Functional Encryption

In defining DDFE, one of our key concerns is generality: we want to achieve for multi-user primitives what Functional Encryption did for single-user primitives. We resist as much as possible the temptation to let the idiosyncrasies of the functionalities we present and implement in this work leak into the definition of DDFE itself. Perhaps the best example of this is in the role of the label. We believe labels, as used in MCFE, are useful for practical use, because in limiting what can be decrypted, they limit data leakage and make it possible to consider using the same primitive over a long time. However, we recognize that some primitives which are of practical use without labels may arise, that some schemes using labels may want to have them interact in more complex ways than perfect matching, and that there is value in our definitions being able to capture existing work. In Section 2.3, we give more details on how our umbrella notion captures a large set of existing primitives, ranging from Public-Key Encryption to Ad Hoc Multi-Input Function Encryption as introduced in Agrawal *et al.* [6].

Definition 1 (Dynamic Decentralized Functional Encryption). *A dynamic decentralized functional encryption scheme over a set of public keys $\mathcal{PK}$ for functionality $\mathcal{F} : \mathcal{L}(\mathcal{PK} \times \mathcal{K}) \times \mathcal{L}(\mathcal{PK} \times \mathcal{M}) \to \{0,1\}^*$ consists of five algorithms:*

- $\mathsf{Setup}(\lambda)$*: Generates and outputs public parameters* pp*. Those parameters are implicit arguments to all the other algorithms;*
- $\mathsf{KeyGen}()$*: Generates and outputs a party's public key* $\mathsf{pk} \in \mathcal{PK}$ *and the corresponding secret key* $\mathsf{sk}_{\mathsf{pk}}$*;*
- $\mathsf{Encrypt}(\mathsf{sk}_{\mathsf{pk}}, m)$*: Takes as input a party's secret key* $\mathsf{sk}_{\mathsf{pk}}$*, a value* $m \in \mathcal{M}$ *to encrypt, and outputs a ciphertext* $\mathsf{ct}_{\mathsf{pk}}$*;*
- $\mathsf{DKeyGen}(\mathsf{sk}_{\mathsf{pk}}, k)$*: Takes as input a party's secret key* sk*, a key space object* k*, and outputs a functional decryption key* $\mathsf{dk}_{\mathsf{pk},k}$*;*
- $\mathsf{Decrypt}\big((\mathsf{dk}_{\mathsf{pk},k_{\mathsf{pk}}})_{\mathsf{pk}\in\mathcal{U}_K}, (\mathsf{ct}_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}_M}\big)$*: Takes as input a finite list of functional decryption keys* $(\mathsf{dk}_{\mathsf{pk},k_{\mathsf{pk}}})_{\mathsf{pk}\in\mathcal{U}_K}$*, a finite list of ciphertexts* $(\mathsf{ct}_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}_M}$*, where* $\mathcal{U}_M, \mathcal{U}_K \in \mathcal{L}(\mathcal{PK})$ *are the lists of senders and receivers, respectively. It outputs a value* $y \in \{0,1\}^*$*.*

We call a DDFE *scheme Public-Key if its encryption algorithm does not make use of the secret key* $\mathsf{sk}_{\mathsf{pk}}$*.*

Correctness: We require that, for all security parameters $\lambda \in \mathbb{N}$*, for all polynomial size lists* $\mathcal{U}_M, \mathcal{U}_K \in \mathcal{L}(\mathcal{PK})$ *of public keys issued by* $\mathsf{KeyGen}()$*,* $(\mathsf{pk}, k_{\mathsf{pk}})_{\mathsf{pk} \in \mathcal{U}_K} \in \mathcal{L}(\mathcal{PK} \times \mathcal{K})$ *and* $(\mathsf{pk}, m_{\mathsf{pk}})_{\mathsf{pk} \in \mathcal{U}_M} \in \mathcal{L}(\mathcal{PK} \times \mathcal{M})$*, it holds that the probability for*

$$\mathsf{Decrypt}((\mathsf{dk}_{\mathsf{pk}, k_{\mathsf{pk}}})_{\mathsf{pk} \in \mathcal{U}_K}, (\mathsf{ct}_{\mathsf{pk}})_{\mathsf{pk} \in \mathcal{U}_M}) = F((\mathsf{pk}, k_{\mathsf{pk}})_{\mathsf{pk} \in \mathcal{U}_K}, (\mathsf{pk}, m_{\mathsf{pk}})_{\mathsf{pk} \in \mathcal{U}_M})$$

is 1, taken over $\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda)$*,* $\mathsf{dk}_{\mathsf{pk}, k_{\mathsf{pk}}} \leftarrow \mathsf{DKeyGen}(\mathsf{sk}_{\mathsf{pk}}, k_{\mathsf{pk}})$ *for all* $\mathsf{pk} \in \mathcal{U}_K$*,* $\mathsf{ct}_{\mathsf{pk}} \leftarrow \mathsf{Encrypt}(\mathsf{sk}_{\mathsf{pk}}, m_{\mathsf{pk}})$ *for all* $\mathsf{pk} \in \mathcal{U}_M$*.*

We stress that each user is identified by a public key pk, which it can generate on its on with the associated secret key, using KeyGen. Anyone can thus dynamically join the system, by publishing its public key.

Remark 2 (Empty keys). Note that, unlike with standard, Single-Input FE, we do not require the empty key ϵ to be in $\mathcal{K}$, because we operate over lists of elements of $\mathcal{PK} \times \mathcal{K}$, so we simply define ϵ as the empty list.
In both Single-Input Functional Encryption and DDFE, the empty key serves to capture all the information about the plaintext that intentionally leaks from every ciphertext (see [11, Section 2]). In Single-Input FE, this is typically only used to highlight the fact that encryption leaks the length of the message.
It is crucial to the security of any Functional Encryption scheme which accepts messages of variable lengths and leaks the length of the message, for otherwise it would be easy to win the `IND` security game by querying $\mathsf{QLeftRight}$ for two messages of different lengths (see Definition 17). With the leakage clearly stated in the functionality of the scheme, such a query would trigger the condition in the game's Finalize, and it would cause the adversary's guess to be discarded.
But in the case of DDFE, more information is usually publicly associated with a ciphertext that simply its length. For instance, the set of users the data should be aggregated with, or the aggregation label, are typically public. Besides, it happens that the leakage of a set of ciphertexts is more than the cumulative leakage of the indidivual ciphertexts. Our AoNE and DSum schemes have this property, and it is expressed by their functionality outputting the relevant information when evaluated on the empty key with a (possibly non-singleton) list of ciphertexts.

Definition 3 (`IND`-Security Game for DDFE). *Let us consider a* DDFE *scheme. No adversary* $\mathcal{A}$ *should be able to win the following security game against a challenger* $\mathcal{C}$*, with unlimited and adaptive access to the oracles* $\mathsf{QNewHonest}$*,* $\mathsf{QEncrypt}$*,* $\mathsf{QLeftRight}$*,* $\mathsf{QDKeyGen}$*, and* $\mathsf{QCorrupt}$ *described below:*

- *Initialize: the challenger* $\mathcal{C}$ *runs the setup algorithm* $\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda)$ *and chooses a random bit* $b \xleftarrow{\$} \{0,1\}$*. It provides* pp *to the adversary* $\mathcal{A}$*;*
- *Participant creation queries* $\mathsf{QNewHonest}$*: the challenger* $\mathcal{C}$ *runs the key generation algorithm* $(\mathsf{pk}, \mathsf{sk}_{\mathsf{pk}}) \leftarrow \mathsf{KeyGen}()$ *to simulate a new participant, stores the association* $(\mathsf{pk}, \mathsf{sk}_{\mathsf{pk}})$ *and returns* pk *to the adversary;*

- *Encryption queries* $\mathsf{QEncrypt}(\mathsf{pk}, m)$*: Recovers the secret key* sk *associated to* pk *and outputs the ciphertext* $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{sk}, m)$*. If* pk *is not associated with any secret key, nothing is returned;*
- *Challenge queries* $\mathsf{QLeftRight}(\mathsf{pk}, m^0, m^1)$*: runs and forwards the output of* $\mathsf{QEncrypt}(\mathsf{pk}, m^b)$*. Wlog. we assume* $m^0 \neq m^1$*.*
- *Functional decryption key queries* $\mathsf{QDKeyGen}(\mathsf{pk}, k)$*: Recovers the secret key* sk *associated to* pk *and outputs the functional decryption key* $\mathsf{dk}_k \leftarrow \mathsf{DKeyGen}(\mathsf{sk}, k)$*. If* pk *is not associated with any secret key, nothing is returned;*
- *Corruption queries* $\mathsf{QCorrupt}(\mathsf{pk})$*: Recovers the secret key* sk *associated to* pk *and outputs it. If* pk *is not associated with any secret key, nothing is returned;*
- *Finalize:* $\mathcal{A}$ *provides its guess* b' *on the bit* b*, and this procedure outputs the result* β *of the security game, according to the analysis given below.*

The output β *of the game depends on some conditions, where* $\mathcal{HS}$ *is the set of honest participants at the end of the game (the set of public keys generated via* $\mathsf{QNewHonest}$*-queries and not corrupted via* $\mathsf{QCorrupt}$*). Finalize outputs the bit* $\beta = (b' = b)$*, unless the following condition (*) is satisfied, in which case Finalize outputs a random bit* β*.*

The condition (*) is true if there exist two lists of public keys $\mathcal{U}_M, \mathcal{U}_K \in \mathcal{L}(\mathcal{PK})$*, two lists of messages* $(\vec{m}^0 = (\mathsf{pk}, m^0_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}_M}, \vec{m}^1 = (\mathsf{pk}, m^1_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}_M})$*, and a list of keys* $\vec{k} = (\mathsf{pk}, k_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}_K}$*, such that* $F(\vec{k}, \vec{m}^0) \neq F(\vec{k}, \vec{m}^1)$*, with*

- $m^0_{\mathsf{pk}} = m^1_{\mathsf{pk}}$*, for all* $\mathsf{pk} \in \mathcal{U}_M$ *such that* $\mathsf{pk} \notin \mathcal{HS}$*;*
- $\mathsf{QLeftRight}(\mathsf{pk}, m^0_{\mathsf{pk}}, m^1_{\mathsf{pk}})$ *or* $\mathsf{QEncrypt}(\mathsf{pk}, m_{\mathsf{pk}})$*-queries have been asked for all* $\mathsf{pk} \in \mathcal{U}_M \cap \mathcal{HS}$*;*
- $\mathsf{QDKeyGen}(\mathsf{pk}, k_{\mathsf{pk}})$*-queries have been asked for all* $\mathsf{pk} \in \mathcal{U}_K \cap \mathcal{HS}$*.*

We say DDFE is IND-secure if for any adversary $\mathcal{A}$*,*

$$\mathsf{Adv}^{\mathtt{IND}}_{\mathsf{DDFE}}(\mathcal{A}) = |2 \times \Pr[\beta = 1] - 1|$$

is negligible.

Intuitively, condition (*) means that the adversary can trivially recover b and win the game, which is thus not a real attack, hence a meaningless output with a random bit. Otherwise, $\beta = 0$ is a wrong guess and $\beta = 1$ is a correct guess during a meaningful attack. As usual, we are interested in adversaries with non-negligible advantage. Note however that the condition of trivial win cannot, in general, be checked in polynomial time. This is because there are exponentially many choices that can be made for the various lists, including the participant public keys and the values of the messages. Even if we impose strict requirements on the functionality, such as the presence of a label and a set of participants, it might not be possible to guarantee that the condition can be checked in polynomial time without a direct analysis of the structure of the functionality. There may exist functionalities for which such a check is a computationally hard problem. The issue of how to efficiently check for violations is thus left to the cryptosystem designers and provers. In the following, we will consider functionalities for which this condition can be efficiently checked.

Now we present several weaker variants of the above security notion.

Definition 4 (sym-IND-Security Game for DDFE). *We define a symmetric-key variant of the above security game in which the Finalize procedure outputs* 0 *if the adversary makes a query of the form* (pk, m_0, m_1) *to* QLeftRight *and queries the same* pk *to* QCorrupt*. This means that the secret key* $\mathsf{sk}_{\mathsf{pk}}$ *not only allows users to encrypt on behalf of party* pk*, but also empowers them to decrypt the ciphertext generated by party* pk*. Thus, the challenge messages* m_0 *and* m_1 *have to be the same to avoid the adversary trivially recovering the random bit* β*. That is, the oracle* QEncrypt *should be used instead of* QLeftRight*.*

Definition 5 (sel-IND-Security Game for DDFE). *We define a selective variant of the above security game in which the adversary is forced to send all its queries to* QNewHonest*, upon which it receives the corresponding public keys. Then it sends all its queries to the oracles* QEncrypt*,* QLeftRight*,* QDKeyGen *and* QCorrupt *in one shot, and receives all of the outputs at once.*

Note that our security notions is strong, in the sense that it allows the adversary to generate malicious public keys on its own. The challenger does not know the corresponding secret keys (which may not exist) for such public keys. More precisely, we allow dishonest key registrations, as originally introduced in [13] in the context of NIKE.

2.3 Versatility of the Notion of DDFE

The notion of DDFE captures many existing primitives. We go over some such primitives and provide details here.

We first show that the notion of public-key encryption is captured by DDFE. That is, we can cast the former as a DDFE for a specific functionality that we present here. Apart from being a warm-up before delving into more advanced primitives, this shows that DDFE is not fundamentally restricted to secret-key primitives.

Public-Key Encryption. Here, the message space $\mathcal{M} = \{0,1\}^* \times \mathcal{PK}$ comprises pairs of plaintext and public keys. The key space is restricted to the identity function over the plaintexts: $\{f_{\mathsf{id}}\}$. The functionality takes as input the list of pairs $(\mathsf{pk}, m_{\mathsf{pk}})$ from all senders $\mathsf{pk} \in \mathcal{U}_M$. In our case, the list $\mathcal{U}_M$ will contain only one user pk_1 who wishes to send the plaintext $\mathsf{pt} \in \{0,1\}^*$ to user pk_2. This information is contained in the message $m_{\mathsf{pk}_1} = (\mathsf{pt}, \mathsf{pk}_2)$.

The functionality also takes the list of pairs $(\mathsf{pk}, k_{\mathsf{pk}})$ from all receivers $\mathsf{pk} \in \mathcal{U}_K$. In our case, the list $\mathcal{U}_K$ only contains the recipient pk'_2. The associated key space object is the identity function f_{id}, which is the only function available here.

The functionality outputs the plaintext if the intended recipient is the actual recipient. That is $F\big((\mathsf{pk}'_2, f_{\mathsf{id}}), (\mathsf{pk}_1, (\mathsf{pt}, \mathsf{pk}_2))\big) = \mathsf{pt}$ if $\mathsf{pk}_2 = \mathsf{pk}'_2$, $\perp$ otherwise.

On any input that does not have that format (for instance on lists $\mathcal{U}_M$ and $\mathcal{U}_K$ of more than one element), the functionality will also output $\perp$.

The above example can be generalized straightforwardly to capture single-input Functional Encryption [11], by considering a larger key space $\{f\}$ that is not only restricted to the identity function.

Decentralized Attribute-Based Encryption. The notion of DDFE can also capture existing decentralized primitives, such as the notion of decentralized Attribute-Based Encryption introduced in [29], as shown below. It also captures the more general Multi-Authority Functional Encryption [15].

Here, the message space $\mathcal{M} = \{0,1\}^* \times \mathcal{P} \times \mathcal{L}(\mathcal{PK})$ comprises tuples, each of which contains a plaintext, a predicate, and a list of public keys. The key space $\mathcal{K} = \mathcal{A} \times \mathcal{ID}$ comprises pairs of an attribute and an identifier.

The functionality takes as input the list of pairs $(\mathsf{pk}, m_{\mathsf{pk}})$ from all senders $\mathsf{pk} \in \mathcal{U}_M$. In our case, the list $\mathcal{U}_M$ will contain only one user pk who wishes to send the plaintext $\mathsf{pt} \in \{0,1\}^*$ to any user with proper credentials, that is, whose attributes satisfy an access policy expressed by a predicate $\mathsf{P} \in \mathcal{P}$. This predicate takes as inputs attributes that are handled by different authorities, listed in $\mathcal{U}$. All of this information is contained in the message $m_{\mathsf{pk}} = (\mathsf{pt}, \mathsf{P}, \mathcal{U})$.

The functionality also takes the list of pairs $(\mathsf{pk}, k_{\mathsf{pk}})$ from all receivers $\mathsf{pk} \in \mathcal{U}_K$. In our case, the list $\mathcal{U}_K$ contains the authorities involved. For each authority, the associated key space object is an attribute, and a global identifier.

The functionality is defined as $F\big((\mathsf{pk}_i, (\mathsf{att}_i, \mathsf{GID}_i))_{\mathsf{pk}_i \in \mathcal{U}_K}, (\mathsf{pk}, (\mathsf{pt}, \mathsf{P}, \mathcal{U}))\big) = \mathsf{pt}$ if $\mathcal{U} = \mathcal{U}_K$, all the identifiers GID_i are the same, and the predicate P on the attributes att_i evaluates to true. If these conditions are not met, or if the input does not have the right syntax (e.g. the list $\mathcal{U}_M$ has more than one element), the functionality outputs $\perp$.

Ad Hoc Multi-Input FE. We now show that DDFE captures more advanced decentralized primitives, such as Ad Hoc Multi-Input FE, introduced in [6].

Here, the message space $\mathcal{M} = \{0,1\}^*$, the key space $\mathcal{K}$ comprises pairs $(f, \mathcal{U})$ where $f : \{0,1\}^\ell \to \{0,1\}^*$ is an ℓ-ary function for arbitrary $\ell \in \mathbb{N}$, and $\mathcal{U}$ is a list of ℓ users.

The functionality takes as input the list of pairs $(\mathsf{pk}, m_{\mathsf{pk}})$ from all senders $\mathsf{pk} \in \mathcal{U}_M$, and the list of pairs $(\mathsf{pk}, k_{\mathsf{pk}})$ from all receivers $\mathsf{pk} \in \mathcal{U}_K$. If all the key space objects agree on a function on the inputs of the list of users $\mathcal{U}_M$, the functionality outputs the evaluation of the function: $F\big((\mathsf{pk}_i, (f_i, \mathcal{U}_i))_{\mathsf{pk}_i \in \mathcal{U}_K}, (\mathsf{pk}_j, m_j)_{\mathsf{pk}_j \in \mathcal{U}_M}\big) = f(m_1, \ldots, m_\ell)$ if $f_i = f$ and $\mathcal{U}_i = \mathcal{U}_M$ for all i, and $|\mathcal{U}_M| = \ell$. It outputs $\perp$ otherwise.

Limitations of DDFE. Whereas the notion of DDFE is a strong generalization of preexisting decentralized variants of Functional Encryption, capturing functionalities not covered by Ad Hoc MIFE or MAFE, it does not cover everything. Function Private [12] and Delegatable [15] variants of Functional Encryption have been introduced, and our definitions leave room for similar variants of DDFE. Some important cryptographic protocols, such as Private Information Retrieval, Oblivious Pseudo Random Functions, or Non-Interactive Key Exchange, similarly cannot be written as DDFE functionalities. DDFE fails to capture key exchange because its definition doesn't allow us to express cryptographic properties of a function evaluation: the idea that the result of an evaluation would "look random" cannot be written as a functionality. It also cannot capture the aforementioned two party interactive protocols because it is non-interactive by nature, while interactivity is a core requirement for PIR and OPRFs, to ensure the protocol is not run more times than any party whishes for.

2.4 DDFE Functionalities

We now give some examples of concrete functionalities. The first two will be of independent interest, but also layers to improve the security and the functionalities of the later Inner-Product DDFE constructions.

All-or-Nothing Encapsulation (AoNE) allows several parties of a group to encapsulate individual messages, that can all be extracted by anybody if and only if all the parties of this group have sent their contributions. Otherwise, the messages remain hidden. The set $\mathcal{U}_M$ of public keys describes the group of parties and the label ℓ imposes a constraint on which encapsulations can be considered together: if for a given pair $(\mathcal{U}_M, \ell)$ all the parties in $\mathcal{U}_M$ send their encapsulations, all the messages can be recovered by anybody, otherwise the messages remain hidden. Note that all the players have to agree on the pair $(\mathcal{U}_M, \ell)$ for their encapsulation, and any encapsulation naturally leaks that pair $(\mathcal{U}_M, \ell)$.

Definition 6 (All-or-Nothing Encapsulation). *AoNE is defined on messages of length L as follows:*

$$\mathcal{K} = \emptyset \qquad\qquad \mathcal{M} = \{0,1\}^L \times \mathcal{S}(\mathcal{PK}) \times \{0,1\}^*.$$

Then, $F(\epsilon, (\mathsf{pk}, (x, \mathcal{U}, \ell))) = (\mathcal{U}, \ell)$ *and*

$$F(\epsilon, (\mathsf{pk}, m_{\mathsf{pk}})_{\mathsf{pk} \in \mathcal{U}_M}) = \begin{cases} (\mathsf{pk}, x_{\mathsf{pk}})_{\mathsf{pk} \in \mathcal{U}_M} & \textit{if condition (*)} \\ \perp & \textit{otherwise.} \end{cases}$$

and AoNE condition (*) is: $\exists \ell \in \{0,1\}^*, \forall \mathsf{pk} \in \mathcal{U}_M, m_{\mathsf{pk}} = (x_{\mathsf{pk}}, \mathcal{U}_M, \ell)$.

Decentralized Sum (DSum) allows several parties of a group to commit to values, so that their sum is automatically revealed when all the parties of this group have sent their contributions. Otherwise, the values remain hidden. The set $\mathcal{U}_M$ of public keys describes the group of parties and the label ℓ imposes a constraint on which values can be added together: if for a given pair $(\mathcal{U}_M, \ell)$ all the parties in $\mathcal{U}_M$ send their values, the sum can be recovered by anybody, otherwise the individual values remain hidden. As above, all the players have to agree on the pair $(\mathcal{U}_M, \ell)$ for their encryption, and any encryption naturally leaks that pair $(\mathcal{U}_M, \ell)$. The terminology *sum* is an abuse, as it works for any Abelian group.

Definition 7 ($(\mathbb{A}, +)$-Decentralized Sum). *DSum is defined for any Abelian group $(\mathbb{A}, +)$ as follows:*

$$\mathcal{K} = \emptyset \qquad\qquad \mathcal{M} = \mathbb{A} \times \mathcal{S}(\mathcal{PK}) \times \{0,1\}^*.$$

Then, $F(\epsilon, (\mathsf{pk}, (x,\mathcal{U}, \ell))) = (\mathcal{U}, \ell)$ and

$$F(\epsilon, (\mathsf{pk}, m_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}_M}) = \begin{cases} \sum_{\mathsf{pk}\in\mathcal{U}_M} x_{\mathsf{pk}} & \text{if condition (*)} \\ \perp & \text{otherwise.} \end{cases}$$

and DSum condition (*) is: $\exists \ell \in \{0,1\}^*, \forall \mathsf{pk} \in \mathcal{U}_M, m_{\mathsf{pk}} = (x_{\mathsf{pk}},\mathcal{U}_M, \ell)$.

Inner-Product DDFE (IP-DDFE). We now present a more advanced functionality for Inner Products. It allows senders with public key pk, as part of a group $\mathcal{U}_M$, to encrypt inputs $\vec{x}_{\mathsf{pk}}$ under a label ℓ. But they maintain control on which computations will be performed on their inputs, as they all have to agree on the weights $\vec{y}_{\mathsf{pk}}$ to produce the functional decryption key that allows the inner-product. The set $\mathcal{U}_M$ of public keys describes the group of parties and the label ℓ imposes a constraint on which values can be aggregated together, the set $\mathcal{U}_K$ of public keys describes the support of the inner-product, and $(\vec{y}_{\mathsf{pk}})_{\mathsf{pk}}$ specifies the weights. If $\mathcal{U}_M = \mathcal{U}_K$ and all the ciphertexts are provided (by all the senders on the same pair $(\mathcal{U}_M, \ell)$), with the appropriate functional decryption key (with the same $(\mathcal{U}_K, (\vec{y}_{\mathsf{pk}})_{\mathsf{pk}})$, one can get the inner-product value, otherwise the individual values remain hidden. As above, all the players have to agree on the pair $(\mathcal{U}_M, \ell)$ for their encryption, and any encryption naturally leaks that pair $(\mathcal{U}_M, \ell)$. Similarly, all the players have to agree on $(\mathcal{U}_K, (\vec{y}_{\mathsf{pk}})_{\mathsf{pk}})$ for the functional decryption key, otherwise they are useless.
Because our construction is based on prime-order groups, we need to impose a bound on the messages and the keys to guarantee that we can perform the discrete logarithm efficiently and recover the result of the functional evaluation in polynomial time.

Definition 8 (Inner-Product DDFE.). *IP-DDFE is defined for a dimension $d \in \mathbb{N}$ and a bound $B \in \mathbb{N}$, and the sets $\mathcal{U}_M$ and $\mathcal{U}_K$ must perfectly match:*

$$\mathcal{K} = \{(\vec{y}_{\mathsf{pk}}, \mathsf{pk})_{\mathsf{pk}\in\mathcal{U}_K} \text{ where } \vec{y}_{\mathsf{pk}} \in [-B, B]^d \text{ and } \mathcal{U}_K \in \mathcal{S}(\mathcal{PK})\}$$
$$\mathcal{M} = [-B, B]^d \times \mathcal{S}(\mathcal{PK}) \times \{0,1\}^*.$$

Then, $F(\epsilon, (\mathsf{pk}, (\vec{x},\mathcal{U}, \ell))) = (\mathcal{U}, \ell)$ and

$$F((\mathsf{pk}, k_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}_K}, (\mathsf{pk}, m_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}_M}) = \begin{cases} \sum_{\mathsf{pk}\in\mathcal{U}_K} \vec{x}_{\mathsf{pk}}^\top \vec{y}_{\mathsf{pk}} & \text{if condition (*)} \\ \perp & \text{otherwise.} \end{cases}$$

and IP-DDFE condition (*) is:

- $\mathcal{U}_K = \mathcal{U}_M$
- $\exists (\vec{y}_{\mathsf{pk}})_{\mathsf{pk} \in \mathcal{U}_K} \in \mathcal{S}([-B, B]^d), \forall \mathsf{pk}' \in \mathcal{U}_K, k_{\mathsf{pk}'} = (\vec{y}_{\mathsf{pk}}, \mathsf{pk})_{\mathsf{pk} \in \mathcal{U}_K}$
- $\exists \ell \in \{0,1\}^*, \forall \mathsf{pk} \in \mathcal{U}_K, m_{\mathsf{pk}} = (\vec{x}_{\mathsf{pk}}, \mathcal{U}_M, \ell)$

We stress that in all the above definition, F should always be understood to be equal to $\perp$ on inputs on which it was not explicitly defined above.

3 Notations and Assumptions

3.1 Groups

Prime Order Groups. We use a prime-order group generator GGen, a probabilistic polynomial time (PPT) algorithm that on input the security parameter 1^λ returns a description $\mathcal{G} = (\mathbb{G}, p, P)$ of an additive cyclic group $\mathbb{G}$ of order p for a 2λ-bit prime p, whose generator is P.

We use implicit representations of group elements as introduced in [21]. For $a \in \mathbb{Z}_p$, define $[a] = aP \in \mathbb{G}$ as the *implicit representation* of a in $\mathbb{G}$. More generally, for a matrix $\mathbf{A} = (a_{ij}) \in \mathbb{Z}_p^{n \times m}$ we define $[\mathbf{A}]$ as the implicit representation of $\mathbf{A}$ in $\mathbb{G}$:

$$[\mathbf{A}] := \begin{pmatrix} a_{11}P & \dots & a_{1m}P \\ \\ a_{n1}P & \dots & a_{nm}P \end{pmatrix} \in \mathbb{G}^{n \times m}$$

We will always use this implicit notation of elements in $\mathbb{G}$, i.e., we let $[a] \in \mathbb{G}$ be an element in $\mathbb{G}$. Note that from a random $[a] \in \mathbb{G}$, it is generally hard to compute the value a (discrete logarithm problem in $\mathbb{G}$). Obviously, given $[a], [b] \in \mathbb{G}$ and a scalar $x \in \mathbb{Z}_p$, one can efficiently compute $[ax] \in \mathbb{G}$ and $[a + b] = [a] + [b] \in \mathbb{G}$.

Pairing-Friendly Groups. We also use a pairing-friendly group generator PGGen, a PPT algorithm that on input 1^λ returns $\mathcal{PG} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, P_1, P_2, e)$, a description of asymmetric pairing-friendly groups where $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are additive cyclic groups of order p for a 2λ-bit prime p, P_1 and P_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable (non-degenerate) bilinear map. Define $P_T := e(P_1, P_2)$, which is a generator of $\mathbb{G}_T$. We again use implicit representation of group elements. For $s \in \{1, 2, T\}$ and $a \in \mathbb{Z}_p$, define $[a]_s = aP_s \in \mathbb{G}_s$ as the implicit representation of a in G_s. Given $[a]_1$, $[b]_2$, one can efficiently compute $[ab]_T$ using the pairing e. For two matrices $\mathbf{A}$, $\mathbf{B}$ with matching dimensions define $e([\mathbf{A}]_1, [\mathbf{B}]_2) := [\mathbf{AB}]_T \in \mathbb{G}_T$.

3.2 Intractability Assumptions

Definition 9 (Computational Diffie-Hellman Assumption). *The CDH assumption states that, in a prime-order group $\mathcal{G} \xleftarrow{\$} \mathsf{GGen}(1^\lambda)$, no PPT adversary can compute $[xy]$, from $[x]$ and $[y]$ for $x, y \xleftarrow{\$} \mathbb{Z}_p$, with non-negligible success probability.*

Equivalently, this assumption states it is hard to compute $[a^2]$ from $[a]$ for $a \xleftarrow{\$} \mathbb{Z}_p$. This comes from the fact that $4[xy] = [(x+y)^2] - [(x-y)^2]$.

Definition 10 (Decisional Diffie-Hellman Assumption). *The DDH assumption states that, in a group $\mathcal{G} \xleftarrow{\$} \mathsf{GGen}(1^\lambda)$, no PPT adversary can distinguish between the two following distributions with non-negligible advantage: $\{([a],[r],[ar]) \mid a,r \xleftarrow{\$} \mathbb{Z}_p\}$ and $\{([a],[r],[s]) \mid a,r,s \xleftarrow{\$} \mathbb{Z}_p\}$.*

Equivalently, this assumption states it is hard to distinguish, knowing $[a]$, a random element from the span of $[\vec{a}]$ for $\vec{a} = \binom{1}{a}$, from a random element in $\mathbb{G}^2$: $[\vec{a}] \cdot r = [\vec{a}r] = \binom{[r]}{[ar]} \approx \binom{[r]}{[s]}$.

Definition 11 (Decisional Bilinear Diffie Hellman Assumption). *The DBDH assumption states that, in a pairing group $\mathcal{PG} \xleftarrow{\$} \mathsf{PGGen}(1^\lambda)$, for any PPT adversary, the following advantage is negligible, where the probability distribution is over $a,b,c,s \xleftarrow{\$} \mathbb{Z}_p$:*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{DBDH}}_{\mathcal{PG}}(\mathcal{A}) = \mid & \Pr[1 \leftarrow \mathcal{A}(\mathcal{PG},[a]_1,[b]_1,[b]_2,[c]_2,[abc]_T)] \\ & - \Pr[1 \leftarrow \mathcal{A}(\mathcal{PG},[a]_1,[b]_1,[b]_2,[c]_2,[s]_T)] \mid.\end{aligned}$$

Definition 12 (Q-fold DBDH). *For any integer Q, the Q-fold DBDH assumption states for any PPT adversary, the following advantage is negligible, where the probability distribution is over $a,b,c_i,s_i \xleftarrow{\$} \mathbb{Z}_p$:*

$$\begin{aligned}\mathsf{Adv}^{Q\text{-}\mathsf{DBDH}}_{\mathcal{PG}}(\mathcal{A}) = \mid & \Pr[1 \leftarrow \mathcal{A}(\mathcal{PG},[a]_1,[b]_1,[b]_2,\{[c_i]_2,[abc_i]_T\}_{i\in[Q]})] \\ & - \Pr[1 \leftarrow \mathcal{A}(\mathcal{PG},[a]_1,[b]_1,[b]_2,\{[c_i]_2,[s_i]_T\}_{i\in[Q]})] \mid.\end{aligned}$$

This Q-fold DBDH assumption is equivalent to classical DBDH assumption:

Lemma 13 (Random Self Reducibility of DBDH). *For any adversary $\mathcal{A}$ against the Q-fold DBDH, running within time t, there exists an adversary $\mathcal{B}$ running within time $t + 2Q(t_{\mathbb{G}_T} + t_{\mathbb{G}_2})$, where $t_{\mathbb{G}_T}$ and $t_{\mathbb{G}_2}$ denote respectively the time for an exponentiation in $\mathbb{G}_T$ and $\mathbb{G}_2$ (we only take into account the time for exponentiations here), such that*

$$\mathsf{Adv}^{Q\text{-}\mathsf{DBDH}}_{\mathcal{PG}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{DBDH}}_{\mathcal{PG}}(\mathcal{B}).$$

Proof. Upon receiving a DBDH challenge $(\mathcal{PG},[a]_1,[b]_1,[b]_2,[c]_2,[s]_T)$, $\mathcal{B}$ samples $\alpha_i, c'_i \xleftarrow{\$} \mathbb{Z}_p$ computes $[c_i]_2 := [\alpha_i \cdot c]_2 + [c'_i]_2$, $[s_i]_T := [\alpha_i \cdot s]_T + [c_i \cdot ab]_T$ for all $i \in [Q]$, and gives the challenge $(\mathcal{PG},[a]_1,[b]_1,[b]_2,\{[c_i]_2,[s_i]_T\}_{i\in[Q]})$ to $\mathcal{A}$. □

3.3 Non-Interactive Key Exchange

We give a definition of Non-Interactive Key Exchange below. This a rephrasing of the m-CKS-heavy model (with dishonest key registrations) as originally introduced in [13] and further refined in [22].

Definition 14 (Non-Interactive Key Exchange). *A NIKE scheme consists of three PPT algorithms:*

- $\mathsf{Setup}(\lambda)$*: Generates and outputs public parameters* pp*. Those parameters are implicit arguments to all the other algorithms;*
- $\mathsf{KeyGen}()$*: Generates and outputs a party's public key* $\mathsf{pk} \in \mathcal{PK}$ *and the corresponding secret key* $\mathsf{sk}_{\mathsf{pk}}$*;*
- $\mathsf{SharedKey}(\mathsf{pk}, \mathsf{sk}_{\mathsf{pk}'})$*: Takes as input a public key and a secret key corresponding to a different public key. Deterministically outputs a shared key* K*.*

Correctness: We require that, for all security parameters $\lambda \in \mathbb{N}$*, it holds that:*

$$\Pr\left[\mathsf{SharedKey}(\mathsf{pk}, \mathsf{sk}_{\mathsf{pk}'}) = \mathsf{SharedKey}(\mathsf{pk}', \mathsf{sk}_{\mathsf{pk}})\right] = 1,$$

where the probability is taken over $\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda)$*,* $(\mathsf{pk}, \mathsf{sk}_{\mathsf{pk}}) \leftarrow \mathsf{KeyGen}()$*,* $(\mathsf{pk}', \mathsf{sk}_{\mathsf{pk}'}) \leftarrow \mathsf{KeyGen}()$*.*

Definition 15 (Security Game for NIKE). *Let us consider a NIKE scheme. No adversary $\mathcal{A}$ should be able to win the following security game against a challenger $\mathcal{C}$, with unlimited and adaptive access to the oracles* $\mathsf{QNewHonest}$*,* $\mathsf{QReveal}$*,* QTest*, and* $\mathsf{QCorrupt}$ *described below:*

- *Initialize: the challenger $\mathcal{C}$ runs the setup algorithm* $\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda)$ *and chooses a random bit* $b \xleftarrow{\$} \{0,1\}$*. It initializes the set $\mathcal{H}$ of honest participants to $\emptyset$. It provides* pp *to the adversary $\mathcal{A}$;*
- *Participant creation queries* $\mathsf{QNewHonest}()$*: the challenger $\mathcal{C}$ runs the* KeyGen *algorithm* $(\mathsf{pk}, \mathsf{sk}_{\mathsf{pk}}) \leftarrow \mathsf{KeyGen}()$ *to simulate a new participant, stores the association* $(\mathsf{pk}, \mathsf{sk}_{\mathsf{pk}})$ *in the set $\mathcal{H}$ of honest keys, and returns* pk *to the adversary;*
- *Reveal queries* $\mathsf{QReveal}(\mathsf{pk}, \mathsf{pk}')$*: Requires that at least one of* pk *and* pk' *be in $\mathcal{H}$. Without loss of generality assume it is* pk*. The challenger returns* $\mathsf{SharedKey}(\mathsf{pk}', \mathsf{sk}_{\mathsf{pk}})$*;*
- *Test queries* $\mathsf{QTest}(\mathsf{pk}, \mathsf{pk}')$*: Requires that both* pk *and* pk' *were generated via* $\mathsf{QNewHonest}$.
  - *If $b = 0$, the challenger returns* $\mathsf{SharedKey}(\mathsf{pk}', \mathsf{sk}_{\mathsf{pk}})$*;*
  - *If $b = 1$, the challenger returns a (uniformly) random value, which it stores so it can consistently answer further queries to* $\mathsf{QTest}(\mathsf{pk}, \mathsf{pk}')$ *or* $\mathsf{QTest}(\mathsf{pk}', \mathsf{pk})$
- *Corruption queries* $\mathsf{QCorrupt}(\mathsf{pk})$*: Recovers the secret key* sk *associated to* pk *from $\mathcal{H}$ and outputs it, then removes the key-pair from $\mathcal{H}$. If* pk *is not associated with any secret key (i.e. it is not in $\mathcal{H}$), then nothing is returned;*
- *Finalize: $\mathcal{A}$ provides its guess b' on the bit b, and this procedure outputs the result β of the security game, according to the analysis given below which aims at preventing trivial wins.*

Finalize outputs the bit $\beta = (b' = b)$ unless a $\mathsf{QCorrupt}$ *query was made for any public key which was involved in a query to* QTest*, or a* $\mathsf{QReveal}$ *query was made for a pair of public keys which was also involved in a* QTest *query, in which case a random bit β is returned.*
We say NIKE *is secure if for any adversary $\mathcal{A}$, the following advantage is negligible:*

$$\mathsf{Adv}_{\mathsf{NIKE}}(\mathcal{A}) = 2 \times |\Pr[\beta = 1] - 1/2|.$$

Definition 16 (Static Security Game for NIKE). *We define a static variant of the security game above in which the adversary does not have access to the* $\mathsf{QCorrupt}$ *oracle, which means all parties created by the challenger will remain honest, and the only corrupt parties are entirely managed by the adversary.*

3.4 Definition of Symmetric Key Encryption

A symmetric key encryption $\mathsf{SKE} = (\mathsf{SEnc}, \mathsf{SDec})$ with key space $\mathcal{K}$ is defined as:

- $\mathsf{SEnc}(K, m)$: given a key K and a message m, outputs a ciphertext ct;
- $\mathsf{SDec}(K, \mathsf{ct})$: given a key K and a ciphertext ct, outputs a plaintext.

Correctness. For all m in the message space and all K in the key space, we must have $\mathsf{SDec}(K, \mathsf{SEnc}(K, m)) = m$.

Security. We say SKE is secure if for any PPT adversary $\mathcal{A}$, the following advantage is negligible:

$$\mathsf{Adv}_{\mathsf{SKE}}(\mathcal{A}) = \left| 2 \times \Pr\left[b' = b : \begin{array}{l} K \overset{\$}{\leftarrow} \mathcal{K}, b \overset{\$}{\leftarrow} \{0,1\} \\ b' \leftarrow \mathcal{A}(1^\lambda)^{\mathsf{QLeftRight}(\cdot,\cdot)} \end{array} \right] - 1 \right|,$$

where the oracle $\mathsf{QLeftRight}$, when queried on m_0, m_1, returns $\mathsf{SEnc}(K, m_b)$.

One-Time Security. We say SKE is One-Time Secure if the above security holds for only one $\mathsf{QLeftRight}$-oracle query. Note that if the key space is larger than the message space, on can simply use the one-time pad to build a One-Time Secure symmetric encryption. Otherwise, a pseudo-random generator can stretch the key to the right length.

3.5 Single-Input Functional Encryption

For some of our constructions, we will need a instatiations of single-input Functional Encryption (for a specific functionalities). A Functional encryption scheme for a family of functions $\mathcal{F}$ consists of the following PPT algorithms:

- $\mathsf{KeyGen}(\lambda)$: on input a security parameter, it outputs a master secret key msk and a public key pk.
- $\mathsf{Encrypt}(\mathsf{pk}, m)$: outputs a ciphertext ct.
- $\mathsf{DKeyGen}(\mathsf{msk}, f)$: on input the master secret key and a function $f \in \mathcal{F}$, it outputs a decryption key dk_f.
- $\mathsf{Dec}(\mathsf{ct}, \mathsf{dk}_f)$: deterministic algorithm that returns a message or a rejection symbol $\perp$ if it fails.

Correctness. For any message m, and any function f in the family $\mathcal{F}$, we have: $\Pr[\mathsf{Dec}(\mathsf{ct}, \mathsf{dk}_f) = f(m)] = 1$, where the probability is taken over $(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{Setup}(\lambda)$, $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{msk}, m)$, and $\mathsf{dk}_f \leftarrow \mathsf{DKeyGen}(\mathsf{msk}, f)$.

Indistinguishability. The security notion is defined by a classical indistinguishability game:

Definition 17 (IND-Security Game for FE). *Let* FE *be a functional encryption scheme. No adversary $\mathcal{A}$ should be able to win the following security game:*

– *Initialize: runs* $(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{Setup}(\lambda)$*, choose a random bit* $b \overset{\$}{\leftarrow} \{0, 1\}$ *and returns* mpk *to* $\mathcal{A}$*.*
– $\mathsf{QLeftRight}(m_0, m_1)$*: on input two messages* (m_0, m_1)*, returns* $\mathsf{Enc}(\mathsf{msk}, m_b)$*.*
– $\mathsf{QDKeyGen}(f)$*: on input a function* $f \in \mathcal{F}$*, returns* $\mathsf{DKeyGen}(\mathsf{msk}, f)$*.*
– *Finalize: from the guess b' of $\mathcal{A}$ on the bit b, it outputs the bit $\beta = (b' = b)$ unless some f was queried to* QDKeyGen *and (m_0, m_1) was queried to* QLeftRight *such that $f(m_0) \neq f(m_1)$, in which case it outputs a uniformly random bit β.*

The adversary $\mathcal{A}$ has unlimited and adaptive access to the left-right encryption oracle QLeftRight*, and to the key generation oracle* QDKeyGen*. We say* FE *is* IND*-secure if for any adversary $\mathcal{A}$,* $\mathsf{Adv}^{\mathit{IND}}_{\mathsf{FE}}(\mathcal{A}) = |2 \times \Pr[\beta = 1] - 1|$ *is negligible.*

We can also define a weaker selective variant, where pairs (m_0, m_1) to QLeftRight-queries are known from the beginning.

Identity-Based Encryption. Here we define the functionality that corresponds to Identity-Based Encryption, originally envisioned in [33], and first realized in [10, 19]. The functionality is described by an identity space $\mathcal{I}$, which can be of exponential size. Each function is described by an identity $\mathsf{id} \in \mathcal{I}$, and given as input a pair (m, id') where m is a payload, and $\mathsf{id}' \in \mathcal{I}$ is an identity, the function outputs m if $\mathsf{id} = \mathsf{id}'$, nothing otherwise.

Inner Product Functionality. For any dimension $d \in \mathbb{N}$ and cyclic group $\mathbb{G}$ of prime order p, the inner product functionality corresponds to the set of functions described by a vector $\vec{y} \in \mathbb{Z}_p^d$ that on input a vector $[\vec{x}] \in \mathbb{G}^d$, outputs $[\vec{x}^\top \vec{y}]$. FE schemes for the inner-product functionality were originally introduced in [3], later in [7] with adaptive security.

We will make use of the following property, satisfied by several FE schemes, including [3, 7]. For concreteness we recall the scheme from [7] in Appendix A.

Property 18 (Linear Homomorphism). An FE for inner products (IP-FE.Setup, IP-FE.Encrypt, IP-FE.DKeyGen, IP-FE.Dec) satisfies the linear homomorphism property if there exists a PPT algorithm Add such that for all $\vec{x}, \vec{x}' \in \mathbb{Z}_p^d$, the following are identically distributed:

$$\Big(\mathsf{IP\text{-}FE.Encrypt}(\mathsf{IP\text{-}FE.pk}, \vec{x}),\ \mathsf{IP\text{-}FE.Encrypt}(\mathsf{IP\text{-}FE.pk}, \vec{x} + \vec{x}')\Big)$$

and

$$\Big(\mathsf{IP\text{-}FE.Encrypt}(\mathsf{IP\text{-}FE.pk}, \vec{x}),\ \mathsf{Add}\Big(\mathsf{IP\text{-}FE.Encrypt}(\mathsf{IP\text{-}FE.pk}, \vec{x}), \vec{x}'\Big)\Big),$$

where $(\mathsf{IP\text{-}FE.pk}, \mathsf{IP\text{-}FE.sk}) \leftarrow \mathsf{IP\text{-}FE.Setup}(\lambda)$.

4 All-or-Nothing Encapsulation from IBE

4.1 Technical Overview

Our generic construction only requires an IBE. Messages are encrypted under the public key of each member of the group successively, using the set of participants $\mathcal{U}_M$ and the label ℓ as the identity. The $|\mathcal{U}_M|$-layers deep encryption is accompanied by the functional decryption key of the IBE for the encrypting participant and the same identity. The only way to recover the messages is to gather all the decryption keys in order to decrypt all layers of IBE encryption: this requires having access to all the ciphertexts. IBE is a well-studied primitive, which admits constructions from multiple hardness assumptions, including pairings [10], LWE [24], or more recently the CDH assumption [20]. This directly implies feasability of AoNE from these assumptions. To keep the size of the ciphertext polynomial in the number of users, we use rate-1 IBE, using hybrid encryption. In Section 5 we give a more efficient construction directly from pairings, inspired by the IBE from [10].

4.2 A Generic Construction of All-or-Nothing Encapsulation

Our construction uses an Identity-Based encryption scheme IBE.

– $\mathsf{Setup}(\lambda)$: Return $pp \leftarrow \mathsf{IBE.Setup}(\lambda)$
– $\mathsf{KeyGen}()$: Return $(\mathsf{pk}, \mathsf{sk}_{\mathsf{pk}}) \leftarrow \mathsf{IBE.KeyGen}()$.
– $\mathsf{Encrypt}(\mathsf{sk}_{\mathsf{pk}}, m)$: Parse $m = (x_{\mathsf{pk}}, \mathcal{U}_M, \ell)$ where $x_{\mathsf{pk}} \in \{0,1\}^L$, $\mathcal{U}_M \in \mathcal{S}(\mathcal{PK})$, and $\ell \in \{0,1\}^*$. If $\mathsf{pk} \notin \mathcal{U}_M$, return $\perp$. Let $n = |\mathcal{U}_M|$ be the cardinal of $\mathcal{U}_M$, and, for some universally accepted order, number the elements in $\mathcal{U}_M$ as $\mathcal{U}_M = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$.
  Let $\alpha_{\mathsf{pk},0} = x_{\mathsf{pk}}$, and for i going from 1 to n, compute
  $$\alpha_{\mathsf{pk},i} := \mathsf{IBE.Encrypt}(\mathsf{pk}_i, (\alpha_{\mathsf{pk},i-1}, \mathcal{U}_M || \ell)).$$
  We write $\alpha_{\mathsf{pk},\mathcal{U}_M,\ell} = \alpha_{\mathsf{pk},n}$ Compute $\gamma_{\mathsf{pk},\mathcal{U}_M,\ell} = \mathsf{IBE.DKeyGen}(\mathsf{sk}_{\mathsf{pk}}, \mathcal{U}_M || \ell)$. Return $(\alpha_{\mathsf{pk},\mathcal{U}_M,\ell}, \gamma_{\mathsf{pk},\mathcal{U}_M,\ell}, \mathcal{U}_M, \ell)$.
– $\mathsf{DKeyGen}(\mathsf{sk}, k)$: There are no keys in this functionality, so no DKeyGen;
– $\mathsf{Decrypt}(\epsilon, (\mathsf{ct}_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}_M})$: Parse the ciphertexts, for all $\mathsf{pk} \in \mathcal{U}_M$, as
  $$\mathsf{ct}_{\mathsf{pk}} = (\alpha_{\mathsf{pk},\mathcal{U}_M,\ell}, \gamma_{\mathsf{pk},\mathcal{U}_M,\ell}, \mathcal{U}_M, \ell),$$
  with common $(\mathcal{U}_M, \ell)$ (otherwise return $\perp$). For each $\mathsf{pk} \in \mathcal{U}_M$, we recover x_{pk} as follows: with $\mathcal{U}_M = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$, recompute the $\alpha_{\mathsf{pk},i}$ for i going from n to 0 as $\alpha_{\mathsf{pk},n} = \alpha_{\mathsf{pk},\mathcal{U}_M,\ell}$ and $\alpha_{\mathsf{pk},i} = \mathsf{IBE.Decrypt}(\gamma_{\mathsf{pk}_i,\mathcal{U}_M,\ell}, \alpha_{\mathsf{pk},i+1})$. Output $(\mathsf{pk}, x_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}_M}$.

Correctness: Correctness follows immediately from the correctness of IBE.

Remark 19 (Rate-1 IBE). To avoid ciphertexts having length exponential in $|\mathcal{U}_M|$, we require that IBE has rate-1 encryption. That is, the ciphertext has the same size as the plaintext plus a polynomial in the security parameter. This can be obtained generically via hybrid encryption: the IBE is used to encrypt a symmetric key, that is used to encrypt the actual message. Assuming such properties of the IBE scheme, our ciphertexts have length linear in $|\mathcal{U}_M|$.

Remark 20. The astute reader will have noticed that the $\gamma_{\mathsf{pk},\mathcal{U}_M,\ell}$ seem to be playing the role of a Functional Decryption Key. Indeed, AoNE could have been defined with keys allowing decryption of the ciphertext if the appropriate key shares (i.e., for that pair $(\mathcal{U}_M, \ell)$) are contributed by all parties. However, our applications of AoNE are such that we would always end up giving out the key share with the corresponding ciphertext, so we gave a definition which is more practical for our uses and may allow constructions in settings where the alternative with keys would be harder to design.

Remark 21. Note that while we show here how to construct AoNE from IBE, it's also possible to construct IBE from AoNE. A possible construction uses only two AoNE identities, one of which creates AoNE ciphertexts that serve as IBE ciphertexts, while the other creates AoNE ciphertexts that serve as IBE functional keys. The secret key for the first identity is made public (it is part of the IBE's public key) while that for the latter remains private. Identities are encoded as labels, and groups are always chosen as the pair of identities. Now to recover the message behind a ciphertext, even generated with the known AoNE secret key of the first identity, one needs an AoNE ciphertext from the second identity for the same label/identity, which effectively acts as an IBE secret key.

4.3 Security Proof

Theorem 22 (IND-Security of AoNE). *The All-or-Nothing Encapsulation scheme described in Section 4.2 is IND-secure (as per Definition 3) assuming the IBE scheme is IND-secure (as per Definition 17).*

The proof can be found in Appendix B.

5 All-or-Nothing Encapsulation from Bilinear Maps

5.1 Technical Overview

This construction is essentially an instantiation of the generic construction given in Section 4.2 using Boneh and Franklin's IBE [10]. However, we make a few optimizations exploiting the structure of the Boneh-Franklin IBE (BF) to achieve short ciphertexts. First, we use the IBE as a Key-Encapsulation Mechanism to generate a symmetric key, which we then use to encrypt the message. Second, we exploit the randomness reusability of El Gamal-like schemes, from which

BF benefits, to only commit to a randomness once. The size difference between the message and the ciphertext in BF comes entirely from that commitment to randomness, so sharing it across all encryptions removes the dependence on the size of the set of participants in the size of the ciphertext.
We provide a direct security analysis of the resulting scheme in Section 5.3.

5.2 A Construction of All-or-Nothing Encapsulation from Bilinear Maps

Our construction uses pairing-friendly groups, a hash function modeled as a random oracle in the security analysis, and a (One-Time Secure) symmetric encryption scheme.

- $\mathsf{Setup}(\lambda)$: Generate $\mathcal{PG} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, P_1, P_2, e) \xleftarrow{\$} \mathsf{PGGen}(1^\lambda)$, a full domain hash function $\mathcal{H}$ from $\{0,1\}^*$ into $\mathbb{G}_1$, and return $\mathsf{pp} = (\mathcal{PG}, \mathcal{H})$. For the sake of clarity, for any input x, we will denote $\mathcal{H}(x) = h_x P_1 = [h_x]_1$, where h_x is the unknown discrete logarithm.
- $\mathsf{KeyGen}()$: Sample $t_{\mathsf{pk}} \xleftarrow{\$} \mathbb{Z}_p$ and return $(\mathsf{pk}, \mathsf{sk}_{\mathsf{pk}}) = ([t_{\mathsf{pk}}]_2, t_{\mathsf{pk}})$.
- $\mathsf{Encrypt}(\mathsf{sk}_{\mathsf{pk}}, m)$: Parse $\mathsf{sk}_{\mathsf{pk}} = t_{\mathsf{pk}} \in \mathbb{Z}_p$ and $m = (x_{\mathsf{pk}}, \mathcal{U}_M, \ell)$ where $x_{\mathsf{pk}} \in \{0,1\}^L$, $\mathcal{U}_M \in \mathcal{S}(\mathcal{PK})$, and $\ell \in \{0,1\}^*$. If $\mathsf{pk} \notin \mathcal{U}_M$, return $\perp$. Otherwise, sample $r_{\mathsf{pk}} \xleftarrow{\$} \mathbb{Z}_p$ and compute the symmetric key $K_{\mathsf{pk},\mathcal{U}_M,\ell}$ as
$$e\left(\mathcal{H}(\mathcal{U}_M||\ell), r_{\mathsf{pk}} \cdot \left(\sum_{\mathsf{pk}' \in \mathcal{U}_M} \mathsf{pk}'\right)\right) = \left[h_{\mathcal{U}_M||\ell} \cdot r_{\mathsf{pk}} \cdot \sum_{\mathsf{pk}' \in \mathcal{U}_M} t_{\mathsf{pk}'}\right]_T,$$
and use it to encrypt x_{pk} as $c_{\mathsf{pk}} = \mathsf{SEnc}(K_{\mathsf{pk},\mathcal{U}_M,\ell}, x_{\mathsf{pk}})$. Compute its share $S_{\mathsf{pk},\mathcal{U}_M,\ell} = t_{\mathsf{pk}} \cdot \mathcal{H}(\mathcal{U}_M||\ell) = [t_{\mathsf{pk}} \cdot h_{\mathcal{U}_M||\ell}]_1$, and output the ciphertext $\mathsf{ct}_{\mathsf{pk}} = (c_{\mathsf{pk}}, [r_{\mathsf{pk}}]_2, S_{\mathsf{pk},\mathcal{U}_M,\ell}, \mathcal{U}_M, \ell)$.
- $\mathsf{DKeyGen}(\mathsf{sk}, k)$: There are no keys in this functionality, so no $\mathsf{DKeyGen}$;
- $\mathsf{Decrypt}(\epsilon, (\mathsf{ct}_{\mathsf{pk}})_{\mathsf{pk} \in \mathcal{U}_M})$: Parse the ciphertexts, for all $\mathsf{pk} \in \mathcal{U}_M$, as $\mathsf{ct}_{\mathsf{pk}} = (c_{\mathsf{pk}}, [r_{\mathsf{pk}}]_2, S_{\mathsf{pk},\mathcal{U}_M,\ell}, \mathcal{U}_M, \ell)$, with common $(\mathcal{U}_M, \ell)$. For each $\mathsf{pk} \in \mathcal{U}_M$, compute
$$K_{\mathsf{pk},\mathcal{U}_M,\ell} = e\left(\sum_{\mathsf{pk}' \in \mathcal{U}_M} S_{\mathsf{pk}',\mathcal{U}_M,\ell}, [r_{\mathsf{pk}}]_2\right) = \left[h_{\mathcal{U}_M||\ell} \cdot r_{\mathsf{pk}} \cdot \sum_{\mathsf{pk}' \in \mathcal{U}_M} t_{\mathsf{pk}'}\right]_T$$
and recover x_{pk} as $x_{\mathsf{pk}} = \mathsf{SDec}(K_{\mathsf{pk},\mathcal{U}_M,\ell}, c_{\mathsf{pk}})$.

Correctness: First, note that the use of $K_{\mathsf{pk},\mathcal{U}_M,\ell}$ is consistent across $\mathsf{Encrypt}$ and $\mathsf{Decrypt}$. Then, the two evaluations correspond to $\left[h_{\mathcal{U}_M||\ell} \cdot r_{\mathsf{pk}} \cdot \sum_{\mathsf{pk}' \in \mathcal{U}_M} t_{\mathsf{pk}'}\right]_T$. Now correctness immediately follows from the correctness of the underlying symmetric encryption scheme.

Remark 23. Note that the sum $\sum_{\mathsf{pk}' \in \mathcal{U}_M} S_{\mathsf{pk}',\mathcal{U}_M,\ell}$ is common to all ciphertexts for the same pair $(\mathcal{U}_M, \ell)$ and can thus be precomputed and reused, such that n messages can be recovered in time $\mathcal{O}(n)$.

5.3 Security Proof

Theorem 24 (IND-Security of AoNE). *The All-or-Nothing Encapsulation scheme described in Section 5.2 is IND-secure (as per Definition 3) under the DBDH assumption, in the random oracle model.*

The proof can be found in the full version [18].

6 Decentralized Sum

6.1 Technical Overview

The starting point of our construction is the "Sum-of-PRFs" technique used by Chase and Chow [16]. The technique aims to enable a set of parties to evaluate local PRFs for a common label ℓ, such that the sum of their local PRFs is zero. It relies on shared seeds between each pair of participants, that are computed on-the-fly using Non-Interactive Key Exchange. Those PRFs can then be added to each participant's input, masking the individual contribution but revealing their sum, because adding the masked ciphertexts causes the PRF evaluation to cancel out.

Remarkably, this is not enough to achieve IND security in the DDFE setting. As such, the random mask would be a deterministic function of the set of participants $\mathcal{U}_M$ and the label ℓ. So, repeated QLeftRight queries to the same pair $(\mathcal{U}_M, \ell)$ with different messages would enable an adversary to break security, simply by subtracting two ciphertexts associated with the same pair $(\mathcal{U}_M, \ell)$ so as to remove the identical masks. This issue can be addressed with a layer of AoNE encryption. Since our AoNE construction is asymmetric and its encryption is randomized, the layer prevents the adversary from combining ciphertexts for the same pair $(\mathcal{U}_M, \ell)$ in a meaningful way. Only when all the ciphertexts are revealed can the adversary remove the AoNE layer, and get access to the underlying ciphertexts. In that case however, the information recovered by the adversary is part of the information revealed by the functionality. For instance, the adversary can subtract two deterministic ciphertexts to obtain the different of the underlying messages. This information can also be learnt by subtracting two sums that are revealed by correctness of the scheme. In general, we show that when the AoNE layer can be removed, the Finalize condition imposes sufficient constraints on the adversary's queries that trivial attacks are no longer on the table.

Moreover, the AoNE layer that lets us achieve full IND security, instead of having to settle for sym-IND security, since, as explained, the AoNE is an asymmetric form of encryption.

6.2 A Generic Construction of Decentralized Sum DDFE for $(\mathbb{A}, +)$

For our construction, we assume a NIKE scheme NIKE, an All-or-Nothing Encapsulation scheme AoNE for messages of length the size of an element of $\mathbb{A}$, and a PRF family $(\mathcal{F}_K)_K$ that takes keys from the NIKE and messages from $\{0,1\}^*$ and outputs pseudo-random elements in $\mathbb{A}$.

- $\mathsf{Setup}(\lambda)$: Run $\mathsf{NIKE.pp} \leftarrow \mathsf{NIKE.Setup}(\lambda)$, $\mathsf{AoNE.pp} \leftarrow \mathsf{AoNE.Setup}(\lambda)$, and output $\mathsf{pp} = (\mathsf{NIKE.pp}, \mathsf{AoNE.pp})$;
- $\mathsf{KeyGen}()$: Run the KeyGen algorithms from the NIKE and the AoNE:

$$(\mathsf{NIKE.pk}, \mathsf{NIKE.sk_{pk}}) \leftarrow \mathsf{NIKE.KeyGen}(),$$

$$(\mathsf{AoNE.pk}, \mathsf{AoNE.sk_{pk}}) \leftarrow \mathsf{AoNE.KeyGen}(),$$

  and output the key pair

$$(\mathsf{pk}, \mathsf{sk_{pk}}) = ((\mathsf{NIKE.pk}, \mathsf{AoNE.pk}), (\mathsf{NIKE.sk_{pk}}, \mathsf{AoNE.sk_{pk}}));$$

- $\mathsf{Encrypt}(\mathsf{sk_{pk}}, m)$: Parse m as $(x, \mathcal{U}_M, \ell)$, with $x \in \mathbb{A}$, $\mathcal{U}_M \in \mathcal{S}(\mathcal{PK})$, and $\ell \in \{0,1\}^*$. Let pk be our encryptor's public key[7]. If $\mathsf{pk} \notin \mathcal{U}_M$, then return $\perp$. Otherwise, for all $\mathsf{pk}' = (\mathsf{NIKE.pk}', \mathsf{AoNE.pk}') \in \mathcal{U}_M$ such that $\mathsf{pk}' \neq \mathsf{pk}$, compute $K_{\mathsf{pk},\mathsf{pk}'} = \mathsf{NIKE.SharedKey}(\mathsf{NIKE.sk_{pk}}, \mathsf{NIKE.pk}')$ and $r_{\mathsf{pk},\mathsf{pk}',\mathcal{U}_M,\ell} = \mathcal{F}_{K_{\mathsf{pk},\mathsf{pk}'}}(\mathcal{U}_M \| \ell)$. Compute $c_{\mathsf{pk}} = x + \sum_{\mathsf{pk}'<\mathsf{pk}} r_{\mathsf{pk},\mathsf{pk}',\mathcal{U}_M,\ell} - \sum_{\mathsf{pk}'>\mathsf{pk}} r_{\mathsf{pk},\mathsf{pk}',\mathcal{U}_M,\ell}$, where the sums are on $\mathsf{pk}' \in \mathcal{U}_M$, on which a total ordering is defined. Return

$$\mathsf{ct_{pk}} = (\mathsf{AoNE.Encrypt}(\mathsf{AoNE.sk_{pk}}, (c_{\mathsf{pk}}, \mathcal{U}_M, \ell)), \mathcal{U}_M, \ell);$$

- $\mathsf{DKeyGen}(\mathsf{sk}, k)$: There are no keys in this functionality, so no $\mathsf{DKeyGen}$;
- $\mathsf{Decrypt}(\epsilon, (\mathsf{ct_{pk}})_{\mathsf{pk}\in\mathcal{U}_M})$: Get $(c_{\mathsf{pk}})_{\mathsf{pk}\in\mathcal{U}_M} = \mathsf{AoNE.Decrypt}(\epsilon, (\mathsf{ct_{pk}})_{\mathsf{pk}\in\mathcal{U}_M})$, and return $\sum_{\mathsf{pk}\in\mathcal{U}_M} c_{\mathsf{pk}}$.

Correctness: The (c_{pk}) should be consistent between $\mathsf{Encrypt}$ and $\mathsf{Decrypt}$ by correctness of AoNE. Besides:

$$\begin{aligned}\sum_{\mathsf{pk}\in\mathcal{U}_M} \mathsf{ct_{pk}} &= \sum_{\mathsf{pk}\in\mathcal{U}_M} \left(x_{\mathsf{pk}} + \sum_{\mathsf{pk}'<\mathsf{pk},\mathsf{pk}'\in\mathcal{U}_M} r_{\mathsf{pk},\mathsf{pk}',\mathcal{U}_M,\ell} - \sum_{\mathsf{pk}'>\mathsf{pk}} r_{\mathsf{pk},\mathsf{pk}',\mathcal{U}_M,\ell} \right) \\ &= \sum_{\mathsf{pk}\in\mathcal{U}_M} x_{\mathsf{pk}} + \sum_{\substack{\mathsf{pk},\mathsf{pk}'\in\mathcal{U}_M \\ \mathsf{pk}'<\mathsf{pk}}} r_{\mathsf{pk},\mathsf{pk}',\mathcal{U}_M,\ell} - r_{\mathsf{pk},\mathsf{pk}',\mathcal{U}_M,\ell} = \sum_{\mathsf{pk}\in\mathcal{U}_M} x_{\mathsf{pk}}\end{aligned}$$

by correctness of NIKE.

6.3 Security Proof

Theorem 25 (IND-Security of DSum). *The Decentralized Sum scheme described in Section 6.2 is IND-secure (as per Definition 3) so long as NIKE is IND-secure (as per Definition 15) and $(\mathcal{F}_K)_K$ is a secure PRF family.*

The proof can be found in the full version [18].

[7] Depending on the details of NIKE and AoNE it may be necessary to explicitly include pk in $\mathsf{sk_{pk}}$ to ensure the following check can be performed.

7 Inner-Product DDFE

7.1 Technical Overview

Our starting point is Chotard *et al.*'s Inner-Product MCFE [17]: as they do, we use a Random Oracle to generate shared randomness across participants for a given label ℓ (in our case a $(\mathcal{U}_M, \ell)$ pair). However, their construction has several drawbacks, which we overcome:

1. Their security game requires that if one ciphertext is queried for a label ℓ, all such ciphertexts must be queried (for the same label ℓ and for all other honest parties). We lift this requirement by protecting ciphertexts with a layer of AoNE.
2. Their Encrypt algorithm is a deterministic function of the message and the label ℓ, and thus they do not tolerate repeated queries to the same participant for the same label. We address this by adding a layer of IP-FE, which randomizes ciphertexts. IP-FE keys are provided in our KeyGen algorithm, and they are protected by an AoNE layer to ensure ciphertexts can only be decrypted once the all the partial functional decryption keys are present.
3. Their scheme, being MCFE, only works in the context of a fixed group. We show how using a PRF to dynamically generate independent secret keys for different groups removes this constraint.
4. To enable non-interactive generation of functional decryption keys in DMCFE, they introduce pairings, and perform message-related operations in $\mathbb{G}_1$ while key-related operations take place in $\mathbb{G}_2$. Instead, we use our DSum to enforce proper key aggregation, which simplifies the scheme to a pairing-free group[8]. DSum has the added benefit that it is a DDFE functionality and thus non-interactive, meaning our Inner-Product scheme is also non-interactive, while their DMCFE has an interactive setup.

7.2 A construction of IP-DDFE

To build our IP-DDFE, we use a cyclic group $\mathbb{G}$ of prime order p where DDH holds, a random oracle $\mathcal{H} : \{0,1\}^* \to \mathbb{G}$, an single-input FE for the inner product functionality, where each function is described by a vector $\vec{y} \in \mathbb{Z}_p^d$, and on input a vector $[\vec{x}] \in \mathbb{G}^d$, outputs $[\vec{x}^\top \vec{y}]$. We require that IP-FE is IND secure, and satisfies Property 18. We also use an All-or-Nothing Encapsulation scheme AoNE, a Distributed Sum DSum over $(\mathbb{Z}_p, +)$, and a PRF family $(\mathcal{F}_K)_K$ that outputs in $\mathbb{Z}_p^d$.

– $\mathsf{Setup}(\lambda)$: Generate $\mathcal{G} = (\mathbb{G}, p, P) \overset{\$}{\leftarrow} \mathsf{GGen}(1^\lambda)$. Generate a full domain hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{G}$. Compute $\mathsf{AoNE.pp} \leftarrow \mathsf{AoNE.Setup}(\lambda)$ and $\mathsf{DSum.pp} \leftarrow \mathsf{DSum.Setup}(\lambda)$. Return:

$$\mathsf{pp} = (\mathcal{G}, \mathcal{H}, \mathsf{NIKE.pp}, \mathsf{AoNE.pp}).$$

[8] Of course, our DSum and our IP-DDFE themselves use AoNE, which may rely on pairings if instantiated with our construction from Section 5.

– $\mathsf{KeyGen}()$: Sample the keys
  - a PRF key K,
  - IP-FE keys $(\mathsf{IP\text{-}FE.pk}, \mathsf{IP\text{-}FE.sk_{pk}}) \leftarrow \mathsf{IP\text{-}FE.KeyGen}(\mathbb{G}, d)$,
  - AoNE keys $(\mathsf{AoNE.pk}, \mathsf{AoNE.sk_{pk}}) \leftarrow \mathsf{AoNE.KeyGen}()$,
  - and DSum keys $(\mathsf{DSum.pk}, \mathsf{DSum.sk_{pk}}) \leftarrow \mathsf{DSum.KeyGen}()$.

  Set the public key $\mathsf{pk} = (\mathsf{IP\text{-}FE.pk}, \mathsf{AoNE.pk}, \mathsf{DSum.pk})$ and the secret key $\mathsf{sk_{pk}} = (K, \mathsf{IP\text{-}FE.sk_{pk}}, \mathsf{AoNE.sk}, \mathsf{DSum.sk})$. Return the key pair $(\mathsf{pk}, \mathsf{sk_{pk}})$.

– $\mathsf{Encrypt}(\mathsf{sk_{pk}}, m)$: Parse m as $(\vec{x}, \mathcal{U}_M, \ell)$, where $\vec{x} \in \mathbb{Z}_p^d$, $\mathcal{U}_M \in \mathcal{S}(\mathcal{PK})$, $\ell \in \{0,1\}^*$, $\vec{s}_{\mathsf{pk},\mathcal{U}_M} = \mathcal{F}_K(\mathcal{U}_M) \in \mathbb{Z}_p^d$, $[h_\ell] = \mathcal{H}(\ell) \in \mathbb{G}$, and

$$c_{\mathsf{pk}} \leftarrow \mathsf{IP\text{-}FE.Encrypt}(\mathsf{IP\text{-}FE.sk_{pk}}, [\vec{x}] + \vec{s}_{\mathsf{pk},\mathcal{U}_M} \cdot [h_\ell]).$$

  Return

$$\mathsf{ct_{pk}} = \big(\mathsf{AoNE.Encrypt}(\mathsf{AoNE.sk_{pk}}, (c_{\mathsf{pk}}, (\mathsf{AoNE.pk}')_{\mathsf{pk}' \in \mathcal{U}_M}, "ct" || \ell)), \mathcal{U}_M, \ell\big)\ ;$$

– $\mathsf{DKeyGen}(\mathsf{sk_{pk}}, k)$: Parse k as $(\vec{y}_{\mathsf{pk}'}, \mathsf{pk}')_{\mathsf{pk}' \in \mathcal{U}_K}$. Compute $\vec{s}_{\mathsf{pk},\mathcal{U}_K} = \mathcal{F}_K(\mathcal{U}_K) \in \mathbb{Z}_p^d$ and

$$d_{\mathsf{pk},k} = \mathsf{DSum.Encrypt}(\mathsf{DSum.sk_{pk}}, (\vec{y}_{\mathsf{pk}}^T \vec{s}_{\mathsf{pk},\mathcal{U}_K}, (\mathsf{DSum.pk}')_{\mathsf{pk}' \in \mathcal{U}_K}, k)).$$

  Compute $d''_{\mathsf{pk},k} = \mathsf{IP\text{-}FE.DKeyGen}(\mathsf{IP\text{-}FE.sk_{pk}}, \vec{y}_{\mathsf{pk}})$ and

$$d'_{\mathsf{pk},k} \leftarrow \mathsf{AoNE.Encrypt}(\mathsf{AoNE.sk_{pk}}, (d''_{\mathsf{pk},k}, (\mathsf{AoNE.pk}')_{\mathsf{pk}' \in \mathcal{U}_K}, "key" || k))$$

  and return $\mathsf{dk}_{\mathsf{pk},k} = (d_{\mathsf{pk},k}, d'_{\mathsf{pk},k})$;

– $\mathsf{Decrypt}((\mathsf{dk}_{\mathsf{pk}',k_{\mathsf{pk}'}})_{\mathsf{pk}' \in \mathcal{U}_K}, (\mathsf{ct_{pk}})_{\mathsf{pk} \in \mathcal{U}_M})$: If $\mathcal{U}_M \neq \mathcal{U}_K$ return $\perp$. Now let $\mathcal{U} = \mathcal{U}_M = \mathcal{U}_K$. Let $k \in \mathcal{K}$ be such that $k = k_{\mathsf{pk}}$ for all $\mathsf{pk} \in \mathcal{U}$. If there is no such k return $\perp$. Parse $\mathsf{dk}_{\mathsf{pk},k}$ as $(d_{\mathsf{pk},k}, d'_{\mathsf{pk},k})$ for all $\mathsf{pk} \in \mathcal{U}$.
  Get

$$(c_{\mathsf{pk}})_{\mathsf{pk} \in \mathcal{U}} = \mathsf{AoNE.Decrypt}(\epsilon, (\mathsf{ct_{pk}})_{\mathsf{pk} \in \mathcal{U}})$$

  and

$$(d''_{\mathsf{pk},k})_{\mathsf{pk} \in \mathcal{U}} = \mathsf{AoNE.Decrypt}(\epsilon, (d'_{\mathsf{pk},k})_{\mathsf{pk} \in \mathcal{U}}).$$

  Then compute $s_k = \sum_{\mathsf{pk} \in \mathcal{U}} \vec{y}_{\mathsf{pk}}^T \vec{s}_{\mathsf{pk},\mathcal{U}} = \mathsf{DSum.Decrypt}(\epsilon, (d_{\mathsf{pk},k})_{\mathsf{pk} \in \mathcal{U}})$.
  For all $\mathsf{pk} \in \mathcal{U}$, compute $z_{\mathsf{pk}} \in \mathbb{G}$ as

$$z_{\mathsf{pk}} \leftarrow \mathsf{IP\text{-}FE.Decrypt}(d''_{\mathsf{pk},k}, c_{\mathsf{pk}}).$$

  Let $\ell \in \{0,1\}^*$ such that all $\mathsf{ct_{pk}}$ for $\mathsf{pk} \in \mathcal{U}$ contain ℓ. If there is no such ℓ, return $\perp$. Otherwise, compute $[h_\ell] = \mathcal{H}(\ell) \in \mathbb{G}$ and return the discrete logarithm in base $[1]$ of

$$\left(\sum_{\mathsf{pk} \in \mathcal{U}} z_{\mathsf{pk}}\right) - s_k \cdot [h_\ell].$$

Correctness: We write $\vec{s}_{\mathsf{pk},\mathcal{U}} = \mathcal{F}_{K_{\mathsf{pk}}}(\mathcal{U})$. By correctness of AoNE, the use of c_{pk} in $\mathsf{Encrypt}$ and in $\mathsf{Decrypt}$ is consistent, as well as the use of $d''_{\mathsf{pk},k}$ in $\mathsf{DKeyGen}$ and $\mathsf{Decrypt}$; By correctness of DSum, we have $s_k = \sum_{\mathsf{pk}\in\mathcal{U}} \vec{y}^T_{\mathsf{pk}} \vec{s}_{\mathsf{pk},\mathcal{U}}$; By correctness property of $\mathsf{IP\text{-}FE}$, we have $z_{\mathsf{pk}} = [\vec{y}^T_{\mathsf{pk}}\vec{x}_{\mathsf{pk}} + \vec{y}^T_{\mathsf{pk}}\vec{s}_{\mathsf{pk},\mathcal{U}} h_\ell]$. Thus we eventually compute and return the discrete logarithm of

$$\left(\sum_{\mathsf{pk}\in\mathcal{U}} z_{\mathsf{pk}}\right) - s_k \cdot [h_\ell] = \left(\sum_{\mathsf{pk}\in\mathcal{U}} [\vec{y}^T_{\mathsf{pk}}\vec{x}_{\mathsf{pk}} + \vec{y}^T_{\mathsf{pk}}\vec{s}_{\mathsf{pk},\mathcal{U}} h_\ell]\right) - \left(\sum_{\mathsf{pk}\in\mathcal{U}} \vec{y}^T_{\mathsf{pk}}\vec{s}_{\mathsf{pk},\mathcal{U}}\right) \cdot [h_\ell]$$
$$= \left[\sum_{\mathsf{pk}\in\mathcal{U}} \vec{y}^T_{\mathsf{pk}}\vec{x}_{\mathsf{pk}} + \vec{y}^T_{\mathsf{pk}}\vec{s}_{\mathsf{pk},\mathcal{U}} h_\ell\right] - \left[\sum_{\mathsf{pk}\in\mathcal{U}} \vec{y}^T_{\mathsf{pk}}\vec{s}_{\mathsf{pk},\mathcal{U}} h_\ell\right] = \left[\sum_{\mathsf{pk}\in\mathcal{U}} \vec{y}^T_{\mathsf{pk}}\vec{x}_{\mathsf{pk}}\right]$$

7.3 Security Proof

Theorem 26 (`sel-sym-IND`-Security of our $\mathsf{IP\text{-}DDFE}$). *The Inner-Product DDFE scheme described in Section 7.2 is `sel-sym-IND`-secure (as per Definition 5) under the DDH assumption, assuming IP-FE is `sel-IND` secure, the AoNE scheme is `sel-sym-IND`-secure, the DSum scheme is `sel-sym-IND`-secure, and $(\mathcal{F}_K)_K$ is a secure PRF family.*

The proof can be found in the full version [18].

Acknowledgments.

This work was supported in part by the European Community's Seventh Framework Programme (FP7/2007-2013 Grant Agreement no. 339563 – CryptoCloud), the European Community's Horizon 2020 Project FENTEC (Grant Agreement no. 780108), the Google PhD fellowship, and the French FUI ANBLIC Project. This work was partially done while the third author was visiting ENS, Paris, and UC Berkeley, California.

References

1. Abdalla, M., Benhamouda, F., Gay, R.: From single-input to multi-client inner-product functional encryption. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019, Part III. LNCS, vol. 11923, pp. 552–582. Springer, Heidelberg (Dec 2019)
2. Abdalla, M., Benhamouda, F., Kohlweiss, M., Waldner, H.: Decentralizing inner-product functional encryption. In: Lin, D., Sako, K. (eds.) PKC 2019, Part II. LNCS, vol. 11443, pp. 128–157. Springer, Heidelberg (Apr 2019)
3. Abdalla, M., Bourse, F., De Caro, A., Pointcheval, D.: Simple functional encryption schemes for inner products. In: Katz, J. (ed.) PKC 2015. LNCS, vol. 9020, pp. 733–751. Springer, Heidelberg (Mar / Apr 2015)

4. Abdalla, M., Catalano, D., Fiore, D., Gay, R., Ursu, B.: Multi-input functional encryption for inner products: Function-hiding realizations and constructions without pairings. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part I. LNCS, vol. 10991, pp. 597–627. Springer, Heidelberg (Aug 2018)
5. Abdalla, M., Gay, R., Raykova, M., Wee, H.: Multi-input inner-product functional encryption from pairings. In: Coron, J., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part I. LNCS, vol. 10210, pp. 601–626. Springer, Heidelberg (Apr / May 2017)
6. Agrawal, S., Clear, M., Frieder, O., Garg, S., O'Neill, A., Thaler, J.: Ad hoc multi-input functional encryption. In: 11th Innovations in Theoretical Computer Science Conference (ITCS 2020). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2020)
7. Agrawal, S., Libert, B., Stehlé, D.: Fully secure functional encryption for inner products, from standard assumptions. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part III. LNCS, vol. 9816, pp. 333–362. Springer, Heidelberg (Aug 2016)
8. Benhamouda, F., Joye, M., Libert, B.: A new framework for privacy-preserving aggregation of time-series data. ACM Trans. Inf. Syst. Secur. 18(3), 10:1–10:21 (2016)
9. Boneh, D., Boyen, X., Goh, E.J.: Hierarchical identity based encryption with constant size ciphertext. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 440–456. Springer, Heidelberg (May 2005)
10. Boneh, D., Franklin, M.K.: Identity-based encryption from the Weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (Aug 2001)
11. Boneh, D., Sahai, A., Waters, B.: Functional encryption: Definitions and challenges. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 253–273. Springer, Heidelberg (Mar 2011)
12. Brakerski, Z., Segev, G.: Function-private functional encryption in the private-key setting. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015, Part II. LNCS, vol. 9015, pp. 306–324. Springer, Heidelberg (Mar 2015)
13. Cash, D., Kiltz, E., Shoup, V.: The twin Diffie-Hellman problem and applications. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 127–145. Springer, Heidelberg (Apr 2008)
14. Chan, T.H.H., Shi, E., Song, D.: Privacy-preserving stream aggregation with fault tolerance. In: Keromytis, A.D. (ed.) FC 2012. LNCS, vol. 7397, pp. 200–214. Springer, Heidelberg (Feb / Mar 2012)
15. Chandran, N., Goyal, V., Jain, A., Sahai, A.: Functional encryption: Decentralised and delegatable. Cryptology ePrint Archive, Report 2015/1017 (2015), http://eprint.iacr.org/2015/1017
16. Chase, M., Chow, S.S.M.: Improving privacy and security in multi-authority attribute-based encryption. In: Al-Shaer, E., Jha, S., Keromytis, A.D. (eds.) ACM CCS 2009. pp. 121–130. ACM Press (Nov 2009)
17. Chotard, J., Dufour Sans, E., Gay, R., Phan, D.H., Pointcheval, D.: Decentralized multi-client functional encryption for inner product. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part II. LNCS, vol. 11273, pp. 703–732. Springer, Heidelberg (Dec 2018)
18. Chotard, J., Dufour-Sans, E., Gay, R., Phan, D.H., Pointcheval, D.: Dynamic decentralized functional encryption. Cryptology ePrint Archive, Report 2020/197 (2020), https://eprint.iacr.org/2020/197
19. Cocks, C.: An identity based encryption scheme based on quadratic residues. In: IMA international conference on cryptography and coding. pp. 360–363. Springer (2001)

20. Döttling, N., Garg, S.: Identity-based encryption from the Diffie-Hellman assumption. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 537–569. Springer, Heidelberg (Aug 2017)
21. Escala, A., Herold, G., Kiltz, E., Ràfols, C., Villar, J.: An algebraic framework for Diffie-Hellman assumptions. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 129–147. Springer, Heidelberg (Aug 2013)
22. Freire, E.S.V., Hofheinz, D., Kiltz, E., Paterson, K.G.: Non-interactive key exchange. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 254–271. Springer, Heidelberg (Feb / Mar 2013)
23. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) 41st ACM STOC. pp. 169–178. ACM Press (May / Jun 2009)
24. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC. pp. 197–206. ACM Press (May 2008)
25. Goldwasser, S., Gordon, S.D., Goyal, V., Jain, A., Katz, J., Liu, F.H., Sahai, A., Shi, E., Zhou, H.S.: Multi-input functional encryption. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 578–602. Springer, Heidelberg (May 2014)
26. Gordon, S.D., Katz, J., Liu, F.H., Shi, E., Zhou, H.S.: Multi-input functional encryption. Cryptology ePrint Archive, Report 2013/774 (2013), http://eprint.iacr.org/2013/774
27. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: Juels, A., Wright, R.N., De Capitani di Vimercati, S. (eds.) ACM CCS 2006. pp. 89–98. ACM Press (Oct / Nov 2006), available as Cryptology ePrint Archive Report 2006/309
28. Joye, M., Libert, B.: A scalable scheme for privacy-preserving aggregation of time-series data. In: Sadeghi, A.R. (ed.) FC 2013. LNCS, vol. 7859, pp. 111–125. Springer, Heidelberg (Apr 2013)
29. Lewko, A.B., Waters, B.: Decentralizing attribute-based encryption. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 568–588. Springer, Heidelberg (May 2011)
30. Libert, B., Titiu, R.: Multi-client functional encryption for linear functions in the standard model from LWE. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019, Part III. LNCS, vol. 11923, pp. 520–551. Springer, Heidelberg (Dec 2019)
31. López-Alt, A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: Karloff, H.J., Pitassi, T. (eds.) 44th ACM STOC. pp. 1219–1234. ACM Press (May 2012)
32. Sahai, A., Waters, B.R.: Fuzzy identity-based encryption. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 457–473. Springer, Heidelberg (May 2005)
33. Shamir, A.: Identity-based cryptosystems and signature schemes. In: Blakley, G.R., Chaum, D. (eds.) CRYPTO'84. LNCS, vol. 196, pp. 47–53. Springer, Heidelberg (Aug 1984)
34. Shi, E., Chan, T.H.H., Rieffel, E.G., Chow, R., Song, D.: Privacy-preserving aggregation of time-series data. In: NDSS 2011. The Internet Society (Feb 2011)

A Single-Input FE for Inner Products

Here we recall the IPFE from [7] on a cyclic group $\mathbb{G}$. Its $\mathtt{IND}$ security is proven in [7], under the DDH assumption in $\mathbb{G}$.

– $\mathsf{IP\text{-}FE.KeyGen}(\mathbb{G}, d \in \mathbb{N})$: $\vec{a} \xleftarrow{\$} \mathsf{DDH}$, $\mathbf{U} \xleftarrow{\$} \mathbb{Z}_p^{d\times 2}$, $\mathsf{pk} = ([\vec{a}], [\mathbf{U}\vec{a}])$, $\mathsf{msk} = \mathbf{U}$. Return $(\mathsf{pk}, \mathsf{msk})$.
– $\mathsf{IP\text{-}FE.Enc}(\mathsf{pk}, \vec{x} \in \mathbb{Z}_p^d)$: $r \xleftarrow{\$} \mathbb{Z}_p$, return $\begin{bmatrix} \vec{a}r \\ \vec{x} + \mathbf{U}\vec{a}r \end{bmatrix} \in \mathbb{G}^{d+2}$.
– $\mathsf{IP\text{-}FE.DKeyGen}(\mathsf{msk}, \vec{y} \in \mathbb{Z}_p^d)$: return $\begin{pmatrix} -\mathbf{U}^\top \vec{y} \\ \vec{y} \end{pmatrix} \in \mathbb{Z}_p^{d+2}$.
– $\mathsf{IP\text{-}FE.Dec}(\mathsf{pk}, [\vec{c}], \vec{k})$: return $[\vec{c}]^\top \vec{k} \in \mathbb{G}$.

B Security Proof: Theorem 22 (IND-Security of AoNE)

The All-or-Nothing Encapsulation scheme described in Section 4.2 is IND*-secure (as per Definition 3) assuming the* IBE *scheme is* IND*-secure (as per Definition 17).*

Proof. Let q_p, q_c denote (polynomial) upper bounds on the number of adversary queries to the QNewHonest oracle, and the number of *unique* pairs $(\mathcal{U}_M, \ell)$ for which the adversary sends at least one QEncrypt or QLeftRight query, respectively. We define the following games for $i \in \{0, \ldots, q_c\}$:

Game $\mathbf{G}_i$: The challenger does as specified in Definition 3, except for queries to QLeftRight. Queries to QLeftRight take as arguments a public key pk and two messages $m_0 = (x_0, \mathcal{U}_{M,0}, \ell_0)$ and $m_1 = (x_1, \mathcal{U}_{M,1}, \ell_1)$. Note that by functionality and by description of the scheme, the response reveals $\mathcal{U}_{M,b}, \ell_b$, so if the adversary wants to avoid the Finalize condition ignoring its guess, it must have $\ell_0 = \ell_1 = \ell$ and $\mathcal{U}_{M,0} = \mathcal{U}_{M,1} = \mathcal{U}_M$. Now let $(\mathcal{U}_{M,j}, \ell_j)$ be the j'th such pair queried to QEncrypt or QLeftRight. In $\mathbf{G}_i$, the challenger will respond to QLeftRight queries by encrypting m_0 if $i < j$ and m_b otherwise, where $b \xleftarrow{\$} \{0, 1\}$ is the random bit chosen by the challenger.

Note that in $\mathbf{G}_0$, all challenge ciphertexts contain the left message, while in $\mathbf{G}_{q_c}$ all challenge ciphertexts contain the right message. Thus we only need to show that $\mathbf{G}_{i-1} \sim_c \mathbf{G}_i$ for all $i \in [q_c]$.

$\mathbf{G}_{i-1} \sim_c \mathbf{G}_i$: We proceed by contradiction, and, from a PPT adversary $\mathcal{A}$ which can distinguish between $\mathbf{G}_{i-1}$ and $\mathbf{G}_i$ with noticeable advantage, we construct a PPT algorithm $\mathcal{B}$ which breaks the IND-security of IBE with noticeable advantage.

$\mathcal{B}$ starts playing the IBE IND-security game and gets a public key IBE.pk. We need to choose the participant whose AoNE public key will be AoNE.pk = IBE.pk carefully, because we wont be able to answer QDKeyGen requests for them. The key is to notice that if the adversary is going to distinguish between $\mathbf{G}_{i-1}$ and $\mathbf{G}_i$, the two need to be different, meaning the adversary $\mathcal{A}$ needs to make at least one query to QLeftRight on $(\mathcal{U}_{M,i}, \ell_i)$ with $x'_0 \neq x'_1$ with noticeable probability, and that, conditioned on that event, $\mathcal{A}$ retains noticeable advantage. We can thus safely assume that $\mathcal{A}$ will make such a query, and abort otherwise. From then on, if it were the case that for every $\mathsf{pk} \in \mathcal{U}_M$, the adversary either makes a

$\mathsf{QEncrypt}$ or $\mathsf{QLeftRight}$ query on $(\mathcal{U}_M, \ell)$ or pk is eventually not honest[9], then the condition in the Finalize part of the security game (see Definition 3) would notice that $x_0' \neq x_1'$ and set the adversary's guess at random, rendering the adversary's efforts fruitless. We can thus safely assume that there is a $\mathsf{pk}^* \in \mathcal{U}_M$ such that pk^* will be created via $\mathsf{QNewHonest}$[10], and the adversary will not query $\mathsf{QEncrypt}$ or $\mathsf{QLeftRight}$ on $(\mathcal{U}_M, \ell)$ for pk^* or query $\mathsf{QCorrupt}$ on pk^*.

We proceed by guessing which query to $\mathsf{QNewHonest}$ will eventually be pk^*. We cannot simply guess a member of $\mathcal{U}_M$ because we do not know anything about $\mathcal{U}_M$ during the initialization phase of the game, and by the time the ith $(\mathcal{U}_M, \ell)$ pair is queried, it is possible that many queries have been made to the $\mathsf{QNewHonest}$ oracle, and at that point it would be too late to embed the IBE public key in the adversary's view. Instead, we guess the index $t^* \in [q_p]$ of the query to $\mathsf{QNewHonest}$ which eventually yields a public key pk_{t^*} which we hope matches pk^*. At that index, we respond with $\mathsf{pk}_{t^*} = \mathsf{IBE.pk}$. Because the adversary will only make polynomially many queries to $\mathsf{QNewHonest}$, our advantage is only polynomially degraded by this guess and the reduction remains valid.

Having done this, we can naturally answer most queries involving pk_{t^*} by using the oracles of the $\mathtt{IND}$ security game of IBE and the fact that our IBE is public key. That is, we answer all $\mathsf{QEncrypt}$ and most (see below) $\mathsf{QLeftRight}$ queries by running $\mathsf{IBE.Encrypt}$ ourselves and making $\mathsf{IBE.QDKeyGen}$ queries.

The exceptions are $\mathsf{QLeftRight}$ queries to any $\mathsf{pk} \in \mathcal{U}_M$ for $(\mathcal{U}_M, \ell)$. Let $n = |\mathcal{U}_M|$ and $\zeta \in [n]$ be such that pk^* is the ζth public key in $\mathcal{U}_M$ for the universally agreed upon order. In responding to $\mathsf{QLeftRight}(\mathsf{pk}, (x_0, \mathcal{U}_M, \ell), (x_1, \mathcal{U}_M, \ell))$, we will compute two sequences of α's as follows: for $s \in \{0, 1\}$, $k \in [\zeta - 1]$, let $\alpha^s_{\mathsf{pk},0} = x_s$ and $\alpha_{\mathsf{pk},k} = \mathsf{IBE.Encrypt}(\mathsf{pk}_k, (\alpha_{\mathsf{pk},k-1}, \mathcal{U}_M||\ell))$. Now compute $\alpha_{\mathsf{pk},\zeta} = \mathsf{IBE.QLeftRight}((\alpha^0_{\mathsf{pk},\zeta-1}, \mathcal{U}_M||\ell), (\alpha^1_{\mathsf{pk},\zeta-1}, \mathcal{U}_M||\ell))$, and compute the rest of the α's and the resulting ciphertext as per $\mathsf{AoNE.Encrypt}$.

When $\mathsf{IBE}.b = 0$, the adversary $\mathcal{A}$ is playing $\mathbf{G}_{i-1}$. When $\mathsf{IBE}.b = 1$, the adversary $\mathcal{A}$ is playing $\mathbf{G}_i$. We only need to check that we do not violate the Finalize condition of the IBE $\mathtt{IND}$-security game. But this is clear because the only $\mathsf{IBE.QLeftRight}$ query we make is for $(\mathcal{U}_M, \ell)$, and for that pair we never get a $\mathsf{AoNE.QLeftRight}$ or $\mathsf{AoNE.QEncrypt}$ query so we never make an $\mathsf{IBE.QDKeyGen}$ query. This concludes our proof. $\square$

[9] Note that here there are two ways for pk to be dishonest: either the adversary has the challenger create pk via $\mathsf{QNewHonest}$ and later corrupts it via $\mathsf{QCorrupt}$, or the adversary generates pk on its own.

[10] Note that here, and in subsequent proofs, we implicitly ignore the very real possibility that the adversary sends a query for a set $\mathcal{U}_M$ for which a *later* query to $\mathsf{QNewHonest}$ generates a $\mathsf{pk} \in \mathcal{U}_M$. Because this happens with negligible probability it is safe to abort when this situation materializes.