Computational Differential Privacy for Encrypted Databases Supporting Linear Queries

Ferran Alborch Escobar[1,2,3], Sébastien Canard[2], Fabien Laguillaumie[3], and Duong Hieu Phan[2]

[1] Applied Crypto Group, Orange Innovation, 14000 Caen, France
ferran.alborch@orange.com
[2] LTCI, Télécom Paris, Institut Polytechnique de Paris, 91120 Palaiseau, France
{sebastien.canard,hieu.phan}@telecom-paris.fr
[3] Univ Montpellier, LIRMM, Montpellier, France
fabien.laguillaumie@lirmm.fr

Abstract. Differential privacy is a fundamental concept for protecting individual privacy in databases while enabling data analysis. Conceptually, it is assumed that the adversary has no direct access to the database, and therefore, encryption is not necessary. However, with the emergence of cloud computing and the «on-cloud» storage of vast databases potentially contributed by multiple parties, it is becoming increasingly necessary to consider the possibility of the adversary having (at least partial) access to sensitive databases. A consequence is that, to protect the on-line database, it is now necessary to employ encryption. At PoPETs'19, it was the first time that the notion of differential privacy was considered for encrypted databases, but only for a limited type of query, namely histograms. Subsequently, a new type of query, summation, was considered at CODASPY'22. These works achieve statistical differential privacy, *by still assuming that the adversary has no access to the encrypted database.*

In this paper, we argue that it is essential to assume that the adversary may eventually access the encrypted data, rendering statistical differential privacy inadequate. Therefore, the appropriate privacy notion for encrypted databases that we use is computational differential privacy, which was introduced by Beimel et al. at CRYPTO '08. In our work, we focus on the case of functional encryption, which is an extensively studied primitive permitting some authorized computation over encrypted data. Technically, we show that any randomized functional encryption scheme that satisfies simulation-based security and differential privacy of the output can achieve computational differential privacy for multiple queries to one database. Our work also extends the summation query to a much broader range of queries, specifically linear queries, by utilizing inner-product functional encryption. Hence, we provide an instantiation for inner-product functionalities by proving its simulation soundness and present a concrete randomized inner-product functional encryption with computational differential privacy against multiple queries. In term of efficiency, our protocol is almost as practical as the underlying inner product functional encryption scheme. As evidence, we provide a full benchmark, based on our concrete implementation for databases with up to 1 000 000 entries. Our work can be considered as a step towards achieving privacy-preserving encrypted databases for a wide range of query types and considering the involvement of multiple database owners.

Keywords: Encrypted Databases · Differential Privacy · Functional Encryption.

1 Introduction

Differential privacy is a data analysis paradigm proposed by Dwork *et al.* in [20,19] to guarantee the privacy of individuals. In broad terms, the objective is to ensure that the presence or not of an individual's data in a database does not significantly impact the results of a data analysis. This is done by blurring the results with some noise, all the while having a precise notion of the trade-off between privacy of the individual and accuracy of the data analysis. To implement this, the concept of *privacy mechanism* is used: a randomized algorithm that takes as input a database and some query and outputs a string. The objective is for this output to be a noisy statistic, where some noise has been added to the real value such that the distributions of the output from the mechanism applied to two databases differing in only one individual are very close. The usefulness of this concept can be seen in the vast amount of academic literature written on the topic (according to the recent survey in [17] over 200 variants of the concept have been proposed so far) and also in real life applications, like the ones by the US Census Bureau [23], Google [22], or Microsoft [18]. Despite the obvious interest in the discipline, there are still some largely unexplored areas, e.g. its interaction with encryption, the potential limitation of an adversary's computational power or even a wider range of privacy mechanisms.

This paradigm was conceptualized to be used in a setting where a database owner who is storing a database wants to release some privacy preserving statistics to untrusted analysts. As such, it was defined as a *statistical* property, i.e., that holds even against a computationally unbounded adversary. However, due to the recent rise in popularity of cloud computing and especially cloud storage of vast databases, this model is today not sufficient to deal with all practical cases. What if the data owner wants to delegate the storage to an external cloud storage service (mainly for cost reasons) while permitting untrusted analysts to make queries?

Differential privacy and encrypted database. To better handle these situations, the concept of encrypted private databases has been introduced by Agarwal *et al.* in [6], where the database owner and the database holder (the cloud storage server) need not be the same entity. In this case, the first step consists, for the data owner, in encrypting its database before sending it to the storage server. Then, when an external analyst wants to ask a query to the database, it negotiates with the database owner for a token which can be used with the storage server to obtain a noisy (differentially private) statistic without further need of the database owner, as can be seen in the diagram of Fig. 1.

However, Agarwal *et al.*, and the subsequent works on the subject [11], rely on standard (statistical) differential privacy arguments, needing to make the assumption that the untrusted analyst has no direct access to the encrypted database. We argue, though, that it is essential to assume that at some point a malicious adversary may have access to such encrypted database, e.g., by colluding with the storage server. Hence, contemplating statistical differential privacy with its computationally unbounded untrusted analyst is inadequate,

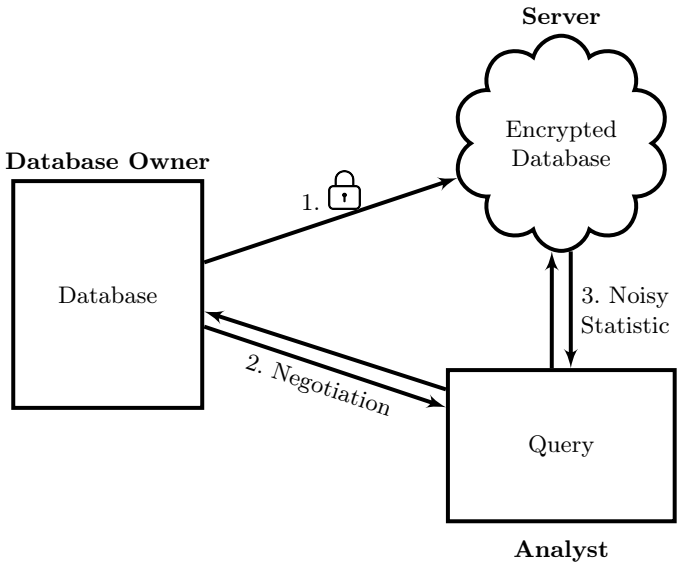

Fig. 1. Diagram of interactions.
A database owner wants to outsource a database to a (honest but curious) server so that private data analysis can still be performed on it. As such, an encrypted database is sent to the server (step 1), and when an analyst wants to perform a query over the database, some negotiation takes place with the database owner which results in a token being received by the analyst (step 2). This token enables the analyst in conjunction with the server to obtain a differentially private noisy statistic (step 3).

given that encryption schemes are proven secure only against computationally bounded adversaries.

Putting both data encryption and differential privacy together, we obtain that data confidentiality *during storage and computation* is provided from the former, while the latter protects the data when *the response of the query is displayed*. Hence, both are necessary to get a full protection of data during all its life cycle. Given this model, statistical DP assumes a statistical adversary which can obviously break any encryption whose security relies on a computational assumption. Therefore, when combining DP with encrypted database, we can at best achieve privacy and security against a computational adversary. If we were to use standard statistical DP, we would need to consider a statistical adversary and then could not restrict the adversary computationally only when attacking the encryption. We have to consider that any potential adversary trying to breach the DP should have the same capabilities when attacking the encrypted database.

Computational Differential Privacy. The concept of computational differential privacy, introduced by Beimel *et al.* in [12] and Mironov *et al.* in [35], has been extensively used in private multi-party computation, i.e., differentially private data analysis over a database owned by more than one entity. The main interest is

that in such setting, it gives much more useful mechanisms than using statistical differential privacy. Indeed, the required noise scales is a much lower order in the multi-party setting [33]. However, in the so-called "client-server" setting, where there is only one database owner, the situation has been less studied, and is not so clear. It was shown by Bun *et al.* in [15] that there exists a task for which there is a computationally differentially private mechanism but any statistically differentially private mechanism will forcibly be inefficient. More recently, Ghazi *et al.* showed in [25] that there exists a non-natural task using strong cryptographic assumptions for which a computationally differentially private mechanism exists but has no statistically differentially private mechanism.

In the domain of computation over encrypted data, three main paradigms exist: multi-party computation (MPC), fully homomorphic encryption (FHE), and functional encryption. MPC addresses the most general form of computation, but it has a shortcoming of requiring a high level of interactions between the parties. This requirement is not practical for specific types of queries on a database. The problem with FHE is that decrypting a ciphertext provides "all or nothing" information. Consequently, when responding to queries from an analyst, the database owner must be the one to recover the encrypted database from the server, decrypt it and compute the corresponding noisy result for each query and send it to the analyst. Then there will need to be interaction between the server and database owner for every query while being no meaningful interaction between the server and the analyst. This violates the requirement to achieve independence for the database owner from interacting with the server for each query from the analyst. For a more detailed discussion see Appendix B. Hence, our idea is to study the case of (randomized) functional encryption, which seems to be the most appropriate in the setting given by Figure 1.

Randomized functional encryption. Functional encryption is a cryptographic concept in which any user in possession of a ciphertext, related to a plain message x, and a functional key sk_f for a function f, can obtain in clear the evaluation $f(x)$. In our context, it's obvious that we need *private* functional encryption as the database owner must manage both the encryption and the key generation processes.

In the case of a randomized functional encryption, the function f could be probabilistic, which permits us, in our setting, to manage the noise inherent to differential privacy. Such possibility was first defined by Alwen *et al.* in [10] and Goyal *et al.* in [27] to extend the concepts of functional encryption towards randomized functionalities. More specifically, a randomized functional encryption scheme takes a description of a randomized (probabilistic) function over a plaintext and randomness space, and generates a functional decryption key. When a ciphertext is decrypted with this functional key an evaluation of the probabilistic function is obtained, with different randomness for different ciphertexts. In the diagram in Fig. 1, what we propose is that the database is encrypted using randomized functional encryption (step 1), and the negotiation between the Database Owner and the Analyst involves the former computing a functional key for the latter (step 2). With this key and a query made to the Server, the

Analyst can obtain statistical information of the database with some differential private noise, using the functional decryption procedure (step 3).

More specifically, Goyal *et al.* in [27] give an instantiation for randomized functional encryption for polynomial-sized circuits which was then used by Garg *et al.* in [24] to construct fully secure functional encryption for all circuits, based on multilinear maps. These works were furthered by Komargodski *et al.* in [31] and Agrawal and Wu in [7] where they give a generic transformation to transform any deterministic functional encryption to a randomized version, the former in the private-key setting and the latter in the public-key setting. Those three works [27,31,7] mention that randomized functional encryption could be used to perform (computational) differentially private analysis on sensitive data, but fail to give a formal analysis of this extension. In this work, we give such analysis.

1.1 Our Contributions

Based on this context, we provide four main contributions in this paper.

1. A new formalization for private functional encryption in the context of computational differential privacy. Firstly, we present a new formalization for private encrypted databases based on functional encryption schemes. Focusing on static databases, we give formal correctness and security notions, taking into account the collusion between a malicious server and a malicious analyst. As far as we know we are the first to consider formal security directly for the collusion between these two entities.

2. A generic result for computational differential privacy in the setting of randomized functional encryption. Secondly, we provide a differentially private mechanism in the context of an encrypted static database which uses a generic randomized functional encryption scheme. We prove that our result is computationally differentially private as long as the used randomized functional encryption scheme satisfies simulation soundness and the scheme instantiates a standard differentially private mechanism. The use of computational differential privacy also allows us to prove this privacy against a possible collusion between the analyst and the server in contrast to previous proposals. We obtain such property by incorporating both the ciphertext and the functional key in the privacy mechanism. This result formalizes and proves the intuition given in [27] about the relation between randomized functional encryption and computational differential privacy.

3. An efficient randomized inner product functional encryption scheme. Thirdly, we give an instantiation for randomized inner product functional encryption, and we prove its simulation soundness. Our construction is based on any generic (deterministic) inner product functional encryption scheme and any distribution guaranteeing statistical differential privacy to the noisy inner product, both used as black-boxes.

4. A computationally differential private encrypted database supporting linear queries. Finally we provide a solution for computationally differentially private encrypted database supporting linear queries, which uses our above randomized inner product functional encryption as a building block. Through our generic result, we prove its computational differential privacy against several inner product queries and collusion between a malicious analyst and server. To the best of our knowledge, this is the first proposal of an encrypted database supporting several inner product differentially private queries and collusion between a malicious analyst and server. We finally provide an implementation of the scheme, proving its practical efficiency for databases with up to 1 000 000 entries.

1.2 Related Works and Comparisons

In regards to encrypted databases from which private data analysis can be performed, to the best of our knowledge, there are only two existing works.

The first encrypted and private database was proposed by Agarwal *et al.* in [6]. They proposed a solution for histogram queries based on several differentially private encrypted counters under continuous observations, one per bin of the histogram. This result is based on the work by Chan *et al.* [16] which is instantiated making use of structured encryption. In general terms, a differentially private counter was instantiated for each of the bins of the histogram and encrypted through homomorphic encryption while the structured encryption scheme is used by the database owner to be able to perform non-noisy data analysis on the database.

The second one is a proposal by Bakas *et al.* in [11] which instantiates a summation of vector coordinates for a database under continuous observation. For that, they consider a multi-input functional encryption scheme allowing different queries, depending on the subset of coordinates being added. In their system, the knowledge of two of their functional keys allows for easy computation of the key for another query without using the master secret key, thus disproving it being a full functional encryption scheme. Regarding the privacy mechanism, they use similar methods to [6] but encrypt only a counting mechanism for each coefficient, without making use of structured encryption.

Compared to those two works, we improve them in three ways, which we now detail. This is also given in Table 1.

1. More important class of queries. In this paper, additionally to our generic result, we give a concrete scheme for linear queries. Linear queries and how to secure databases under them has been a well studied subject [39,32].

Definition 1 (Equation 3, [39]). *A* linear query *for some data elements* $x_1, \ldots, x_k$ *has the form*

$$q(x_1, \ldots, x_k) = \sum_{i=0}^{k} a_i \cdot x_i$$

for some $k > 1$ *and fixed query weights* a_i*.*

Table 1. Comparison with related works on differential privacy in encrypted databases.

Proposal	Query type	Access to database (via collusion)	DP type
[6]	Histogram	✗	Statistical
[11]	Summation	✗	Statistical
Our work	Inner product	✓	Computational

The most notable cases are predicate counting queries (e.g. histograms, marginal queries and group-by queries) and weighted sum queries (e.g. weighted averages, differences and evaluation of linear regression models).

2. Encrypted databases and/or encrypted mechanisms. In the seminal work by Agarwal et al. [6], the basic concept they deal with is as follows: a database is hidden and an analyst can ask queries, receiving a differentially private response, while the database owner can still perform non-noisy queries to the database. To instantiate such properties for private histogram queries they make use of both a structured encryption scheme and a set of encrypted differentially private counters. The overall idea is that the database is encrypted through a structured encryption scheme through which the database owner can perform non-noisy queries, while the histograms are instantiated through an encrypted differentially private counter for each of the bins. The analyst can only access the latter and not directly the encrypted database. This means that there are both an encrypted database and an encrypted private mechanism at the same time, and the different entities have access to different objects, depending on their rights.

In [11] the option of getting rid of the encrypted database is explored. In essence, for each coefficient of the vector they encrypt a slightly more elaborated counter than in [6], through the use of an encryption scheme with certain homomorphic properties. Then, when an analyst asked for a key to compute the summation of a subset of coefficients, the sent key is the sum of the respective private keys. This means their solution is based only on an encrypted private mechanism. It follows that the ability for the database owner to perform non-noisy queries to the database is lost.

It is also worthy of note that in [27] an instantiation of a randomized encryption for polynomial-sized circuits is given, based on indistinguishability obfuscation. In their discussion the authors say that their results extend directly to differential privacy, but without giving any formal treatment of it. Their case would also be in the side of having only an encrypted mechanism, since non-noisy statistics may not be queried to the ciphertexts. This is also due since the setting in which they contemplate is different to ours, since the server has the master secret key and acts as an intermediary between the database owners and the analyst.

The direction taken in this work is to explore the possibility of getting rid of the encrypted private mechanisms and leaving only an encrypted database where the mechanism is "baked into" the encryption scheme. Hence, we allow

both the data owner and the analyst to make queries on it. We consider that this is relevant in a real-life application.

The overall idea is to have an encrypted database with which several different tokens related to a query can be used to extract the desired output. Ideally this can be used to obtain both differentially private and non-noisy responses depending on the token applied, and our inner product construction based on (randomized) functional encryption satisfies this. The use of functional encryption instead of structured encryption is due to the need of a building block capable of supporting randomized functionalities, and furthermore opens the door to extending the mechanism to several different databases being queried privately with the same token in the future.

But this decision makes the differential privacy analysis slightly more delicate than in previous works due to mainly two reasons. First of all, the tokens received by the adversary are now more relevant and should be directly included in the differential privacy analysis. This means that parts of the encryption scheme are being studied through the differential privacy analysis: the classical statistical differential privacy no longer makes sense, given that the security definitions of encryption schemes necessarily rely on probabilistic polynomial time adversaries. Thus we decided to use computational differential privacy instead, as introduced in [12]. Secondly, the previous works essentially use ℓ independent private mechanisms, one for each "independent" query they allow. This allows them to extend privacy for one query to several directly. However, we are considering our different queries as forming part of the same mechanism, and as such must make it explicit in our differential privacy analysis.

Multiple queries in the context of differential private mechanism is something that has already been studied in the literature, for the non-encrypted case. In such case, it is necessary to take of the noise that is used so that nobody can, query after query, remove it. Such existing work consider either adding noise proportional to a query index [40], or managing the noise accordingly [29]. For linear queries, this can simply be thwarted by only accepting linearly independent queries. In any case, our work does not introduce any new issue regarding this multiple queries case, and any of the above method can obviously be adapted to our result.

3. Managing a collusion between server and analyst. As mentioned before, differential privacy originated as a means to ensure the privacy on an individual while performing data analysis over a database. The general idea is to compute the (exact) statistic and then add some carefully chosen noise such that the result is precise enough to be useful but such that the distributions obtained from databases with and without the data of an individual respectively are very close. In this setting, it makes sense to consider this property as statistical, or in other words, that it holds against a computationally unbounded analyst to cover all bases.

However, when considering our setting of private *encrypted* databases, one must consider the possibility of the encrypted database eventually falling in the hands of a malicious entity. The noteworthy difference from the original

setting of differential privacy is that, as we have mentioned before, the security of encryption schemes is usually only guaranteed against PPT adversaries. As such, analyzing the privacy of our setting when the malicious entity has access to the actual ciphertexts (as is the case of a collusion between a malicious analyst and a malicious server) against a computationally unbounded entity is no longer adequate. This leads us to argue that computational differential privacy should be used to analyze such private encrypted databases.

More specifically, in our case we analyze a non-obvious mechanism. The previous works consider the mechanism as only the output obtained by the analyst when using a token with the server. However, we consider our mechanism to be the output together with the token and the ciphertexts, and prove that even having access to all these outputs simultaneously, a PPT analyst cannot break (computational) differential privacy. Since all the information in possession of both the analyst and the server is contained in the output of this mechanism, we succeed in proving differential privacy against a collusion between them.

1.3 Organisation

In Section 2 we give the formalizations necessary to follow our results. In Section 3 we give the generic result of computational differential privacy for randomized functional encryption. In Section 4 we present our randomized inner product instantiation and prove it correct and secure. In Section 5 we present our proposal for computationallly differentially private encrypted database supporting linear queries and in Section 6 we give some more concrete results the implementation of our proposals. Finally, in Section 7 we give our conclusions and a frame for future works.

2 Formalizations

In this section we will recall the classical definitions and introduce our new definitions.

2.1 Notations

One-dimensional elements (such as those in $\mathcal{X}$, $\mathbb{Z}$, $\mathbb{G}$...) will be noted as lower-case letters ($x, y, \ldots$), while multi-dimensional elements (such as those in $\mathcal{X}^\ell$, $\mathbb{Z}^\ell$, $\mathbb{G}^\ell$...) will use bold lower-case letters ($\boldsymbol{x}, \boldsymbol{y}, \ldots$). For a natural number $q > 0$ we denote as $[q]$ the set $\{1, \ldots, q\}$. Let D be a probability distribution, $x \leftarrow D$ means the element x is sampled from the distribution D, while for any set $\mathcal{Y}$, $y \xleftarrow{\$} \mathcal{Y}$ means that y is sampled uniformly at random from $\mathcal{Y}$. Finally, a function f is said to be *negligible* over n ($f = \text{negl}(n)$) if for all $k \in \mathbb{N}_{>0}$, there exists $n_0 \in \mathbb{N}_{>0}$ such that for any $n > n_0$ then $|f(n)| < 1/n^k$.

2.2 Differential Privacy

Differential privacy is a private data mechanism property first proposed by Dwork *et al.* in [20,19] as a way to guarantee in a precise manner the privacy for the data of an individual in a pool of data. In broad terms, the way this is ensured is by adding some noise to the statistic computed over the dataset in such a way that the probability of getting the same result with two different databases (one with the individual's data and one without) is essentially the same. This notion of privacy soon became the main paradigm, and more than 200 variants have been defined since then, as by the survey made by Desfontaines and Pejó in [17].

A basic concept needed to properly define differential privacy is that of neighbourhood between databases. This concept specifically delineates what is the difference in the database between adding an individual's data or not. In our case we will say that two databases are *neighbouring* if their ℓ_1 distance is at most one. The standard definition for this property is (ϵ, δ)-differential privacy for static databases as stated below. In this work we will constrain ourselves to the study of static databases. From this section onward we will consider $\mathcal{X}$ to be a database space, $\mathcal{R}$ to be a randomness space, $\mathcal{S}$ to be an output space contained in the multidimensional real numbers and $\mathcal{F}$ a family of deterministic functions $f : \mathcal{X} \to \mathcal{S}$ representing the queries to obtain the plain statistics.

Definition 2 (Adapted from Definition 2.4, [21]). *Let ϵ, δ be two real numbers. A randomized algorithm for $f \in \mathcal{F}$, $\mathcal{M}_f : \mathcal{X} \times \mathcal{R} \to \mathcal{S}$ is* (ϵ, δ)-differential private *((ϵ, δ)-DP) if for all $S \subseteq \mathcal{S}$, every pair of neighbouring databases $x, x' \in \mathcal{X}$ and $r, r' \leftarrow \mathcal{R}$*

$$\Pr\left[\mathcal{M}_f(x; r) \in S\right] \leq e^{\epsilon} \cdot \Pr\left[\mathcal{M}_f(x'; r') \in S\right] + \delta.$$

There are two slight changes from Definition 2.4 in [21]. First of all, we have adapted the definition of randomized function to be in line with the standard in randomized functional encryption, and as such have added the specific randomness seed as an input. Secondly, we have explicitly added which query $f \in \mathcal{F}$ the mechanism $\mathcal{M}$ is protecting. This is also for ease of notation further down the line, when considering several different queries and relating to the key generation in the randomized functional encryption scheme.

Note that this definition handles only one query at a time, and we would be interested in the property for Q queries. However, since the output space $\mathcal{S}$ must be contained on the multidimensional reals, one can consider the query space as $\mathcal{F}^Q$ and each query for the mechanism as the conjunction of Q queries. That way this definition allows for analysis for multiple queries.

The previous definition is statistical, in the sense that it takes into account the distribution of all possible outputs, however, when trying to combine them with cryptographic concepts, we find that this would correspond to playing against computationally unbounded adversaries. As such, combination of these more statistical (and more standard) variants of differential privacy with well-known cryptographic primitives and methods proves to be sometimes unfeasible and/or

Table 2. Experiment b for (Q, ϵ_κ)-IND-CDP.

Experiment b :
1: $(x_0, x_1, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\kappa)$ with $x_0, x_1 \in \mathcal{X}$ neighbouring
2: $\tilde{b} \leftarrow \mathcal{A}_2^{\mathcal{O}(x_b,\cdot,\cdot)}(\mathsf{st})$
Output: $\tilde{b}$

unrealistic. Therefore it is natural to consider relaxations on the definition of differential privacy to allow for bounding the adversary to being computationally efficient. This is what Mironov *et al.* proposed in [35], which we adapt to our needs.

Definition 3 (Adapted from Definition 3, [35]). *Let $\kappa \in \mathbb{N}$ be a security parameter, Q be an integer and $\mathcal{M} : \mathcal{X} \times \mathcal{F} \times \mathcal{R} \to \mathcal{S}$ a randomized algorithm. Then, for a stateful* PPT *algorithm $\mathcal{A}$ the attack game works as follows. The challenger $\mathcal{C}$ selects a bit $b \xleftarrow{\$} \{0,1\}$ and proceeds with experiment b (Table 2) where the oracle $\mathcal{O}(x_b, \cdot)$ denotes the evaluation of the mechanism $\mathcal{M}(x_b, \cdot; r)$ for some $r \leftarrow \mathcal{R}$.*

*We say that $\mathcal{M}$ provides (Q, ϵ_κ)-*indistinguishable computational differential privacy *((Q, ϵ_κ)-IND-CDP) if there exists a negligible function* $\mathrm{negl}(\cdot)$ *such that for any $\mathcal{A}$ limited to accessing $\mathcal{O}$ Q times*

$$\Pr\left[\tilde{b} = 1 | b = 0\right] \leq e^{\epsilon_\kappa} \cdot \Pr\left[\tilde{b} = 1 | b = 1\right] + \mathrm{negl}(\kappa).$$

There are several changes from Definition 3 in [35]. The two clearer ones are as we discussed with Definition 2, where we have adopted the standards for randomized functional encryption. A part from this, we also consider a PPT adversary instead of a Turing machine with polynomial sized advice string since, as mentioned before, the computational power of the adversary needs to be the same when considering privacy as when considering security. Finally, despite the fact that we could consider the query space as $\mathcal{F}^Q$ to expand to handling Q queries, however, we are interested in allowing for adaptivity in the choice of query for the adversary. As such, we have considered a stateful adversary.

There is one important thing to note about this definition. Were the adversary $\mathcal{A}$ to be computationally unbounded, then this only says that for any fixed κ the mechanism $\mathcal{M}$ is $(\epsilon_\kappa, \delta_\kappa)$-DP for a negligible δ_κ and any set of Q queries. This means that any mechanism that satisfies $(\epsilon_\kappa, \delta_\kappa)$-DP for all sets of Q queries and δ_κ negligible on κ, will also satisfy ϵ_κ-IND-CDP.

Note that to prove a one-query mechanism adaptive for several queries in statistical differential privacy, the property of composability is required, which is not known if it holds for computational differential privacy. However, since our generic result ties the adaptive computational differential privacy of the whole mechanism understood as the whole randomized functional encryption scheme to the adaptive computational privacy of only the output, as long as the output satisfies the adaptivity, the property transfers to the whole mechanism. Therefore, if we choose an output distribution for the scheme that satisfies statistical

differential privacy (and therefore composability) our mechanism will be private against an analyst choosing queries adaptively.

Finally, two additional key concepts, that we recall below, need to be taken into account when considering the efficiency of concrete constructions for differential privacy: the sensitivity, which is used to measure accurately the size of the noise needed to privatize a specific query; and the utility of a given mechanism, which essentially tells how close the noisy statistic will be to the expected non-noisy value.

Definition 4 (Adapted from Definition 3.1, [21]). *Let $x, x' \in \mathcal{X}$ be two neighbouring databases. The* ℓ_1-sensitivity *of a function f is*

$$\Delta_f := \max_{\|x-x'\|_1=1} \|f(x) - f(x')\|_1.$$

This can be naturally extended to the ℓ_1-sensitivity of a family of queries by taking the maximum over the family of queries.

Definition 5 (Adapted from Definition 2.4, [16]). *Let $\mathcal{M} : \mathcal{X} \times \mathcal{F} \times \mathcal{R} \to \mathcal{S}$, $\mathcal{M}(x, f; r) = f(x) + e(r)$ for some database space $\mathcal{X}$, response space $\mathcal{S}$, function space $\mathcal{F} : \mathcal{X} \to \mathcal{S}$ and randomness space $\mathcal{R}$ be a differentially private mechanism. We say $\mathcal{M}$ is* (α, δ)-useful *if*

$$\Pr\left[|\mathcal{M}(x, f; r) - f(x)| \leq \alpha\right] \geq 1 - \delta$$

for any $x \in \mathcal{X}$, $f, \in \mathcal{F}$ and $r \leftarrow \mathcal{R}$.

2.3 Randomized Functional Encryption

Given that our objective is to mix functional encryption with differential privacy (which inherently uses randomness to blur the information) it is clear that we need to introduce some randomness into the functional encryption. To do so, we will follow the paradigm set by Goyal *et al.* in [27] for general randomized functional encryption. In this section we will consider $\mathcal{X}$ to be a database space, $\mathcal{R}$ to be a randomness space, $\mathcal{S}$ to be an output space and $\mathcal{F}$ a family of randomized (probabilistic) functions $f : \mathcal{X} \times \mathcal{R} \to \mathcal{S}$, where $r \in \mathcal{R}$ is understood as the seed for the probabilistic sampling of the randomized function f and as such, as true randomness completely unknown to the adversary. The reason for this is to be able to ensure that for any database $x \in \mathcal{X}$, $f(x; r)$ is computed always with the same random seed, otherwise the database could be leaked by sampling the random function with multiple different seeds. We will focus on the secret key variant of randomized functional encryption, which is defined as follows.

Definition 6 (Adapted from Section 2, [27]). *Let $\kappa \in \mathbb{N}_{>0}$ be a security parameter. We define a* secret-key randomized functional encryption scheme *supporting the family of randomized functions $\mathcal{F}$ the following tuple of* PPT *algorithms:*

- $\mathsf{SetUp}(1^\kappa, \mathcal{F})$*: given the security parameter and family of functions as an input, it outputs some public parameters* param *and a master secret key* msk*. We will assume the public parameters as inputs in all other algorithms.*
- $\mathsf{Enc}(\mathsf{msk}, x)$*: given the master secret key* msk *and some plaintext* $x \in \mathcal{X}$ *as inputs, it outputs a ciphertext* c_x*.*
- $\mathsf{KeyGen}(\mathsf{msk}, f)$*: given the master secret key* msk *and a description of the randomized function* $f \in \mathcal{F}$ *as inputs, it outputs a functional key* sk_f*.*
- $\mathsf{Dec}(c_x, sk_f)$*: a deterministic algorithm that given a ciphertext* c_x *and a functional key* sk_f *as inputs, it outputs a string* s*.*

There is a correctness notion linked to this definition of encryption scheme, however it is not as straightforward as in standard functional encryption due to the randomization of the output. Because of this, we need to assure the computational indistinguishability of the output string from the Dec algorithm with the functionality output. Our definition is as follows.

Definition 7. *Let* $\kappa \in \mathbb{N}_{>0}$ *be a security parameter and* $\mathsf{RFE} = (\mathsf{SetUp}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ *be a secret-key randomized functional encryption scheme supporting the family of randomized functions* $\mathcal{F}$*. We say it is* correct *if for any plaintext* x *and any set of functions* $f^1, \ldots, f^Q \in \mathcal{F}$ *the following distributions are computationally indistinguishable:*

- $\boldsymbol{Real(1^\kappa, \mathcal{F})} := \{s^i \leftarrow \mathsf{Dec}(c_x, sk_{f^i})\}_{i\in[Q]}$*, where*
  $(\mathsf{param}, \mathsf{msk}) \leftarrow \mathsf{SetUp}(1^\kappa)$
  $c_x \leftarrow \mathsf{Enc}(\mathsf{msk}, x)$.
  $sk_{f^i} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f^i)$ *for all* $i \in [Q]$.
- $\boldsymbol{Ideal(1^\kappa, \mathcal{F})} := \{f^i(x; r^i)\}_{i\in[Q]}$ *where* $r^i \leftarrow \mathcal{R}$.

This definition differs from the one in [27] because it is stated for only one plaintext instead of several. The difference lies in the fact that their constructions are both for several simultaneous plaintexts and in the public key setting so the adversary can obtain as many ciphertexts as it wants. By considering several plaintexts in the definition it ensures that the randomness in the output is distinct for different ciphertexts and different functional keys. More specifically, it is required that for one same functional key, different ciphertexts give different outputs and viceversa, which ensures that the randomness in the output comes from both the randomness in the encryption and key generation. Also note that simply having some randomness in both encryption and key generation does not satisfy the condition since they could simply not be used for the randomness in the output. The encryption scheme of our proposal in Section 4 is an example of this behaviour.

However, in this work we are interested in the case of randomized functional encryption as a means of constructing a differentially private mechanism supporting several queries to one database and we are in the secret key setting. As such, it makes sense to consider this relaxation of the definition so as to allow solutions which do not change the noise for different ciphertexts, since we will only be considering one.

The same distinction between selective and adaptive adversaries can be done in the randomized setting as in the deterministic one, as well as both the indistinguishability and simulation based security and their non-equivalence. For the purpose of this work we have slightly changed the simulation-based security definition to add a condition on how the key generator simulator works. In our case we are interested in the particular case of randomized functional encryption schemes implementing differential private data analysis, and there are some schemes that satisfy the definition in [27] that very clearly will not be differentially private. For example, let $\mathsf{FE} = (\mathsf{SetUp}^{\mathsf{FE}}, \mathsf{Enc}^{\mathsf{FE}}, \mathsf{KeyGen}^{\mathsf{FE}}, \mathsf{Dec}^{\mathsf{FE}})$ be a functional encryption scheme and D be a probability distribution. We define a simple randomized functional encryption scheme $\mathsf{RFE} = (\mathsf{SetUp}^{\mathsf{RFE}}, \mathsf{Enc}^{\mathsf{RFE}}, \mathsf{KeyGen}^{\mathsf{RFE}}, \mathsf{Dec}^{\mathsf{RFE}})$ as follows:

- $\mathbf{SetUp}^{\mathbf{RFE}}(1^\kappa, \mathcal{F})$:
  $(\mathsf{msk}^{\mathsf{FE}}, param^{\mathsf{FE}}) \leftarrow \mathsf{SetUp}^{\mathsf{FE}}(1^\kappa)$
  $Output\ (\mathsf{msk}^{\mathsf{RFE}}, param^{\mathsf{RFE}}) = (\mathsf{msk}^{\mathsf{FE}}, param^{\mathsf{FE}})$
- $\mathbf{Enc}^{\mathbf{RFE}}(\mathbf{msk}^{\mathbf{RFE}}, \boldsymbol{x})$:
  $c_x \leftarrow \mathsf{Enc}^{\mathsf{FE}}(\mathsf{msk}^{\mathsf{FE}}, x)$
  $Output\ c_x$
- $\mathbf{KeyGen}^{\mathbf{RFE}}(\mathbf{msk}^{\mathbf{RFE}}, \boldsymbol{f})$:
  $e_f \leftarrow D$
  $sk_f \leftarrow \mathsf{KeyGen}^{\mathsf{FE}}(\mathsf{msk}^{\mathsf{FE}}, f)$
  $Output\ sk_f^{\mathsf{RFE}} = (e_f, sk_{\hat{f}})$
- $\mathbf{Dec}^{\mathbf{RFE}}(\boldsymbol{c_x}, \boldsymbol{sk_f^{\mathbf{RFE}}})$:
  $f(x) \leftarrow \mathsf{Dec}^{\mathsf{FE}}(c_x, sk_f)$
  $s \leftarrow f(x) + e_f$
  $Output\ s$

It is clear that this scheme cannot be differentially private since the noise is given out in the functional key. However, if FE is simulation sound against one ciphertext, this scheme will also be simulation secure against one ciphertext following the definition in [27]. By substituting the FE algorithms for their respective simulators we get the simulators for the RFE scheme. More details can be found in Appendix C.

This situation means that the definition given in [27] is not enough to characterize randomized functional encryption for differential privacy: we need a stronger definition. To obtain it, we make use of a characteristic of the standard definition, where the key generation simulator algorithm has access to the ideal functionality $\mathsf{KeyIdeal}$. This ideal functionality takes as input a randomized function g and for every x_i in the challenge it outputs $v_i^g = g(x_i; r^i)$ with some chosen randomness r^i from the randomness space.

For our new stronger definition, we will ask an extra requirement from the key generation simulator algorithm: the simulator to query the $\mathsf{KeyIdeal}$ and to output a simulated functional key sk_g^* which should satisfy that for any simulated ciphertext c_i^* the decryption algorithm's output is $\mathsf{Dec}(c_i^*, sk_g^*) = v_i^g$. This allows us to discard the trivial schemes since without having access to the challenges x_i, the key generation simulator algorithm cannot recover the noise to be

Table 3. Real and ideal experiments in 1-SEL-SIM security for RFE.

$\underline{\mathsf{Exp}^{real}_{\mathcal{A}}(1^\kappa, \mathcal{F})}$
1: $(x, \mathsf{st}_1) \leftarrow \mathcal{A}_1(1^\kappa, \mathcal{F})$ where $x \in \mathcal{X}$
2: $(\mathsf{param}, \mathsf{msk}) \leftarrow \mathsf{SetUp}(1^\kappa, \mathcal{F})$
3: $c_x \leftarrow \mathsf{Enc}(x, \mathsf{msk})$
4: $\gamma \leftarrow \mathcal{A}_2^{\mathcal{O}_1(\mathsf{msk}, \cdot)}(c_x, \mathsf{st}_1)$
Output: $(x, \{f\}, \{sk_f\}, \gamma)$

$\underline{\mathsf{Exp}^{ideal}_{\mathcal{A},\mathsf{Sim}}(1^\kappa, \mathcal{F})}$
1: $(x, \mathsf{st}_1) \leftarrow \mathcal{A}_1(1^\kappa, \mathcal{F})$ where $x \in \mathcal{X}$
2: $(\mathsf{param}, c_x^*, \mathsf{st}') \leftarrow \mathsf{EncSim}(1^\kappa, \mathcal{F})$
3: $\gamma \leftarrow \mathcal{A}_2^{\mathcal{O}'_1(\mathsf{st}', \cdot)}(c_x^*, \mathsf{st}_1)$
Output: $(x, \{f'\}, \{sk^*_{f'}\}, \gamma)$

output in the functional key. More details can be found in Appendix C. Furthermore, we show that our instantiation for randomized inner product satisfies this requirement so it is not an unattainable condition. We have also adapted the definition to the secret key setting.

Definition 8. *Let* $\kappa \in \mathbb{N}_{>0}$ *be a security parameter and* RFE $=$ (SetUp, Enc, KeyGen, Dec) *be a secret-key randomized functional encryption scheme for the randomized function family* $\mathcal{F}$*. We say* RFE *is* 1-SEL-SIM-secure *if there exists a* PPT *simulator* Sim $=$ (EncSim, KeyGenSim) *such that for every* PPT *adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, the outputs of the real and ideal experiments (see Table 3) are computationally indistinguishable, where the oracles are described as follows.*

1. ***Real Experiment:*** $\mathcal{O}_1(\mathsf{msk}, \cdot)$ *refers to the non-simulated key generation oracle* $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$*. The set* $\{f\}$ *denotes the key queries made by* $\mathcal{A}_2$*.*
2. ***Ideal experiment:*** $\mathcal{O}'_1(\mathsf{st}', \cdot)$ *denotes the simulated key generation algorithm* $\mathsf{KeyGenSim}(\mathsf{st}', \cdot)$ *that has oracle access to the ideal functionality* $\mathsf{KeyIdeal}(x, \cdot)$*. The functionality* KeyIdeal *accepts key queries* f' *and returns* $v^{f'} = f'(x; r)$ *for some chosen randomness* $r \leftarrow \mathcal{R}$*. We require that for simulated ciphertext* c_x^* *and simulated key* $sk^*_{f'}$*, the decryption value is as such* $Dec(c_x^*, sk^*_{f'}) = v^{f'}$*. The set* $\{f'\}$ *denotes the queries made by* KeyGenSim *to* KeyIdeal*.*

In this definition we only consider the case for one challenge ciphertext, since that is all we need for our results. However, the "strengthening" of the definition is easily extendable to several ciphertexts. Also note that for our definition we do not consider a decryption oracle where the adversary can input a ciphertext and a function to obtain the function applied to the ciphertext. This is due to the fact that it will allow us to prove simulation soundness for a greater amount of instantiations.

2.4 Private Functional Encryption

The end goal is to instantiate a private encrypted database supporting linear queries. Agarwal *et al.* gave in [6] a formalization as to what properties such an object should satisfy. In their formalization they consider them as private structured encryption schemes for dynamic databases, in this work we adapt their paradigm to private functional encryption scheme for static databases. Let us give the definition for which we take Definition 4.1 in [6] as a reference.

Definition 9. *Let $\kappa \in \mathbb{N}_{>0}$ be a security parameter, $\epsilon > 0$ a privacy parameter and $Q \in \mathbb{N}_{>0}$ a positive integer. We define a* private functional encryption scheme for static databases *supporting the family of queries $\mathcal{F}$ with error distribution D_ϵ as the following tuple of polynomial time protocols:*

- $\mathsf{SetUp}_{\boldsymbol{DO},\boldsymbol{S},\boldsymbol{A}}((1^\kappa, 1^\epsilon, x); \perp; (f^1, \ldots, f^Q))$*: is a three-party protocol involving the database owner* $\boldsymbol{DO}$*, server* $\boldsymbol{S}$ *and analyst* $\boldsymbol{A}$*. The database owner inputs the security and privacy parameter κ, ϵ as well as the database x, the server inputs nothing and the analyst inputs the set of Q queries they want to ask $f_1, \ldots, f_Q \in \mathcal{F}$. As output, the database owner receives a master secret key* msk*, the server receives an encrypted database c_x and the analyst receives a set of functional keys $sk_{f^1}, \ldots, sk_{f^Q}$. Everyone receives a set of parameters* param*.*
- $\mathsf{EQuery}_{\boldsymbol{DO},\boldsymbol{S}}((\mathsf{msk}, g); c_x)$*: is a two-party protocol involving the databases owner* $\boldsymbol{DO}$ *and the server* $\boldsymbol{S}$*. The database owner inputs the master secret key* msk *and a query $g \in \mathcal{F}$, while the server inputs the encrypted database c_x. As output, the database owner receives a response s and the server receives nothing.*
- $\mathsf{PQuery}_{\boldsymbol{A},\boldsymbol{S}}(sk_{f^i}; c_x)$*: is a two-party protocol between the analyst* $\boldsymbol{A}$ *and the server* $\boldsymbol{S}$*. The analyst inputs a functional decryption key sk_{f^i}, while the server inputs the encrypted database c_x. As output, both receive the response s^i.*

Note that in this definition, to keep in line with the definition from [6] we have decided to put the queries asked as an input to the setup phase. This way the setup remains a three-party protocol between all the entities and the private query a two-party protocol between the analyst and the server. Despite this, the functional encryption paradigm offers us more flexibility and another way of conceiving the protocols may be considered. For example, by allowing the analyst adaptivity on their query requests the setup phase becomes a two-party protocol between the database owner and the analyst and the private query phase becomes a concatenation of two two-party protocols, one between the analyst and the database owner and one between the analyst and the server. This alternative definition may seem more appropriate for some cases and makes full use of the adaptivity for computational differential privacy in Definition 3.

As usual, this new object needs a correctness definition where, differently than in [6] we consider that both types of queries should be considered in this analysis. Our definition is based on Definition 4.2 in [6].

Definition 10. *Let $\kappa \in \mathbb{N}_{>0}$ be a security parameter, $\epsilon > 0$ a privacy parameter, $Q \in \mathbb{N}_{>0}$ a positive integer and $\mathsf{PFE} = (\mathsf{SetUp}, \mathsf{EQuery}, \mathsf{PQuery})$ be a private functional encryption scheme for static databases supporting the family of functions $\mathcal{F}$ with error distribution D_ϵ. We say it is* correct *if for any database x and any set of queries $f^1, \ldots, f^Q \in \mathcal{F}$ the following distributions are computationally indistinguishable:*

- $\mathbf{Real}(\mathbf{1^\kappa}, \mathbf{1^\epsilon}) := \{s^i \leftarrow \mathsf{PQuery}_{\boldsymbol{A},\boldsymbol{S}}(sk_{f^i}; c_x)\}_{i \in [Q]}$ *where,* $(\mathsf{msk}; c_x; (sk_{f^1}, \ldots, sk_{f^Q})) \leftarrow \mathsf{SetUp}_{\boldsymbol{DO},\boldsymbol{S},\boldsymbol{A}}((1^\kappa, 1^\epsilon, x); \perp; (f^1, \ldots, f^Q))$.

– $\mathbf{Ideal(1^\kappa, 1^\epsilon)} := \{f^i(x) + e^i\}_{i\in[Q]}$ *where* $e^i \leftarrow D_\epsilon$

and for any database x *and query* $f \in \mathcal{F}$ *the following probability holds*

$$\Pr\left[s \leftarrow \mathsf{EQuery}_{\boldsymbol{DO},\boldsymbol{S}}((\mathsf{msk}, f); c_x) \neq f(x)\right] = negl(\kappa)$$

where the probability is taken over $(\mathsf{msk}; c_x; (sk_{f^1}, \ldots, sk_{f^Q})) \leftarrow \mathsf{SetUp}((1^\kappa, 1^\epsilon, x); \perp; (f^1, \ldots, f^Q))$.

Finally we need to discuss the security notion. In [6] they describe three types of adversary: persistent, statistical and snapshot. The first one refers to an adversary corrupting permanently the server, the second one refers to an adversary corrupting the analyst and the third one refers to an adversary corrupting the server at only one point in time. A security definition for each one of them is given, however, the possible collusions between these adversaries are not formally handled. We argue that by giving one single security definition considering the case of collusion between a malicious server and analyst contains all the rest of individual cases, and therefore proving this last security implies all the rest.

Definition 11. *Let* $\kappa \in \mathbb{N}_{>0}$ *be a security parameter,* $\epsilon > 0$ *a privacy parameter,* $Q \in \mathbb{N}_{>0}$ *a positive integer and* $\mathsf{PFE} = (\mathsf{SetUp}, \mathsf{EQuery}, \mathsf{PQuery})$ *a private functional encryption scheme for static databases supporting the family of queries* $\mathcal{F}$ *with error distribution* D_ϵ*. We denote* $f^1, \ldots, f^Q \in \mathcal{F}$ *as* F*. We say* PFE *is* 1-database-secure and private *if there exists a* PPT *simulator* $\mathsf{Sim} = (\mathsf{SetUpSim}, \mathsf{EQuerySim})$ *such that for every* PPT *adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *corrupting both the server* $\boldsymbol{S}$ *and analyst* $\boldsymbol{A}$ *the outputs of the real and ideal experiments (see Table 4) are computationally indistinguishable, where the oracles are described as follows:*

– ***Real Experiment:*** $\mathcal{O}_1$ *refers to the non-simulated encrypted query protocol* $\mathsf{EQuery}_{\boldsymbol{DO},\boldsymbol{S}}(\mathsf{msk}, \cdot; \cdot)$*. The set* $\{f\}$ *denotes the queries asked to this oracle,* $\mathcal{O}_2$ *refers to the private query protocol* $\mathsf{PQuery}_{\boldsymbol{A},\boldsymbol{S}}(\cdot; \cdot)$*. The set* $\{s\}$ *denotes the responses of* $\mathcal{O}_2$*,*

– ***Ideal Experiment:*** $\mathcal{O}'_1$ *refers to the simulated encrypted query protocol* $\mathsf{EQuerySim}_{\boldsymbol{DO},\boldsymbol{S}}(\mathsf{st}', \cdot; \cdot)$*. The set* $\{f'\}$ *denotes the queries made to this oracle,* $\mathcal{O}'_2$ *refers to the private query protocol* $\mathsf{PQuery}_{\boldsymbol{A},\boldsymbol{S}}(\cdot; \cdot)$*. The set* $\{s'\}$ *denotes the responses of* $\mathcal{O}'_2$*,*

and the mechanism $\mathcal{M}$ *defined as follows*

$$\mathcal{M}(x, F; r) = \begin{cases} c_x \leftarrow \mathsf{SetUp}^r_{\boldsymbol{DO},\boldsymbol{S},\boldsymbol{A}}((1^\kappa, 1^\epsilon, x); \perp; F) \\ sk_{\hat{f}^i} \leftarrow \mathsf{SetUp}^r_{\boldsymbol{DO},\boldsymbol{S},\boldsymbol{A}}((1^\kappa, 1^\epsilon, x); \perp; F) \\ s \leftarrow \mathsf{PQuery}_{\boldsymbol{A},\boldsymbol{S}}(sk_{f^i}; c_x) \end{cases} \tag{1}$$

satisfies (Q, ϵ)-IND-CDP.

3 CDP for Randomized Functional Encryption

In this section we provide our results for a differentially private mechanism supporting randomized functional encryption for the private queries in encrypted databases.

Table 4. Real and ideal experiments in 1-database security for PFE.

$\mathsf{Exp}^{real}_{\mathcal{A}}(1^\kappa, 1^\epsilon)$
1: $(x, F, \mathsf{st}_1) \leftarrow \mathcal{A}_1(1^\kappa)$ where $x \in \mathcal{X}$ and $F \in \mathcal{F}^Q$
2: $(\mathsf{msk}; c_x, sk_{f^1}, \ldots, sk_{f^Q}) \leftarrow \mathsf{SetUp}_{\mathcal{C},\mathcal{A}}(1^\kappa, 1^\epsilon, x; F)$
3: $\gamma \leftarrow \mathcal{A}_2^{\mathcal{O}_1(\mathsf{msk},\cdot,\cdot),\mathcal{O}_2(\cdot,\cdot)}(c_x, \mathsf{st}_1)$
Output: $(x, \{f\}, \{s\}, \gamma)$

$\mathsf{Exp}^{ideal}_{\mathcal{A},\mathsf{Sim}}(1^\kappa)$
1: $(x, F, \mathsf{st}_1) \leftarrow \mathcal{A}_1(1^\kappa)$ where $x \in \mathcal{X}$ and $F \in \mathcal{F}^Q$
2: $(\mathsf{st}'; c^*_x, sk^*_{f^1}, \ldots, sk^*_{f^Q}) \leftarrow \mathsf{SetUpSim}_{\mathcal{C},\mathcal{A}}(1^\kappa, 1^\epsilon; F)$
3: $\gamma \leftarrow \mathcal{A}_2^{\mathcal{O}'_1(\mathsf{st}',\cdot,\cdot),\mathcal{O}'_2(\cdot,\cdot)}(c^*_x, \mathsf{st}_1)$
Output: $(x, \{f'\}, \{s'\}, \gamma)$

3.1 Overview

Following the notation of Fig. 1, we would have that the encrypted database is c_x in the form of a functional encryption ciphertext (step 1); the negotiation between the database owner and the analyst (step 2) consists in the analyst sending a function f and the database owner responding with the functional key $sk_{\hat{f}}$; and the noisy statistic (step 3) consists on a string of the form $f(x) + e_f$, computed by the server with c_x and $sk_{\hat{f}}$, using the decryption procedure of the functional encryption.

More precisely, let $\mathcal{M}'$ be a classical differentially private mechanism such that for a plain database x and a function f it outputs the value $f(x)+e_f$ for e_f sampled from some distribution D_ϵ that renders the mechanism statistically differentially private. Our idea is to then obtain a randomized functional encryption $\mathsf{RFE} = (\mathsf{SetUp}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ such that the output of the decryption algorithm $\mathsf{Dec}(c_x, sk_{\hat{f}})$ is distributed as $\hat{f}(x) = f(x) + D_\epsilon$, where $\mathsf{msk} \leftarrow \mathsf{SetUp}(1^\kappa)$, $c_x \leftarrow \mathsf{Enc}(\mathsf{msk}, x)$ and $sk_{\hat{f}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \hat{f})$.

The next step is to properly define the privacy mechanism $\mathcal{M}$ that most accurately represents our problem. It is clear that the mechanism must incorporate the noisy statistic during both the decryption and the functional key processes, since both outputs are received by the analyst at some point during the interaction. If the server was to be trusted, those two values would suffice as a DP mechanism. However, since the objective is to deal with an honest but curious server, the database encryption process should also be considered in the privacy mechanism. Therefore, when looking at all three steps, considering the three values (ciphertext, functional key and noisy plaintext) within the DP mechanism, we are capable to cover for a collusion between the server and the analyst. As such, our DP mechanism $\mathcal{M}$ on input a database x and query f outputs $c_x, sk_{\hat{f}}$ and $\mathsf{Dec}(c_x, sk_{\hat{f}})$. Note that the value $\mathsf{Dec}(c_x, sk_{\hat{f}})$ is redundant since it can be computed from c_x and $sk_{\hat{f}}$. However, we put it in to keep coherence with the analysis given in Figure 1 and to emphasize the information available through both the analyst (the functional key) and the server (encrypted data).

Given this DP mechanism, it is obvious that the standard statistical definition of differential privacy is no longer adequate, since an adversary with unlimited power can always break the underlying encryption and get all the information about the database. Therefore our proof below is done using the above defined (Q, ϵ_κ)-IND-CDP (see Definition 3), hence using the simulators of our new security definition for randomized functional encryption (see Definition 8).

3.2 Formal Analysis

Let us start the formal analysis. Let $\mathcal{F}$ be the family of deterministic functions such that $\forall f \in \mathcal{F}$, $f : \mathcal{X} \to \mathcal{S}$. Let $\mathcal{R}$ be a set of random values and let $e : \mathcal{R} \to \mathcal{S}$ be a sample generation function over a pre-defined distribution D_ϵ with values in $\mathcal{S}$. From that, we define the family $\hat{\mathcal{F}}$ of randomized functions such that $\forall \hat{f} \in \hat{\mathcal{F}}$, $\hat{f} : \mathcal{X} \times \mathcal{R} \to \mathcal{S}$ and $\hat{f}(x; r) = f(x) + e(r)$, where $r \leftarrow \mathcal{R}$ is used as a seed to sample $e(r) \leftarrow D_\epsilon$. Such definition next permits us to formally define mechanism $\mathcal{M}' : \mathcal{X} \times \mathcal{F} \times \mathcal{R} \to \mathcal{S}$ such that $\mathcal{M}'(x, f; r) = \hat{f}(x; r)$. This corresponds to a classical DP mechanism for a function $f \in \mathcal{F}$. Our purpose in this section is to generically transform it into an equivalent DP mechanism for an encrypted database.

For this purpose, we consider $\mathsf{RFE} = (\mathsf{SetUp}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ as a secure secret-key randomized functional encryption scheme for the family of randomized functions $\hat{\mathcal{F}}$. Therefore, using the structure given in Figure 1, and based on the notions and notations given in Section 2, we define the following.

1. The encrypted database corresponds to $c_x \leftarrow \mathsf{Enc}(\mathsf{msk}, x)$ where $x \in \mathcal{X}$ is a plain database, and where $(\mathsf{param}, \mathsf{msk}) \leftarrow \mathsf{SetUp}(1^\kappa, \hat{\mathcal{F}})$ as previously been executed once for all by the database owner;
2. the negotiation is done for a set of queries represented as functions $\hat{f}^1, \ldots, \hat{f}^Q \in \hat{\mathcal{F}}$ and gives, for all $i \in [Q]$, $sk_{\hat{f}^i} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \hat{f}^i)$;
3. the noisy statistic phase executes, for all $i \in [Q]$, $s^i \leftarrow \mathsf{Dec}(c_x, sk_{f_i})$ which is obtained by the analyst.

From the correctness of the used RFE, we obtain that $\forall i \in [Q]$, s_i is computationally indistinguishable from the value $\hat{f}^i(x; r_i) = f^i(x) + e(r_i)$, where $r_i \leftarrow \mathcal{R}$.

Let us now focus on our new DP mechanism $\mathcal{M}$ for an encrypted database. As we consider that both the server and the analyst could be corrupted. Such a DP mechanism must include all the information available to both, namely c_x, $sk_{\hat{f}}$ and $\mathsf{Dec}(c_x, sk_{\hat{f}})$ for one specific query. But we now need to take care of the used randomness, and be more precise on where it should be put. To be as generic as possible, we consider that the randomness space is divided into two parts. Hence, $\mathcal{R} = \mathcal{R}_x \times \mathcal{R}_f$ and $\forall r \in \mathcal{R}$, r can be written as $r = (r_x, r_f)$, where $r_x \in \mathcal{R}_x$ (resp. $r_f \in \mathcal{R}_f$) is the seed for the randomness sampled in the encryption (resp. key generation) algorithm to compute c_x (resp. $sk_{\hat{f}}$). Then our DP mechanism for encrypted database $\mathcal{M} : \mathcal{X} \times \mathcal{F} \times \mathcal{R} \to \mathcal{S}$ is defined as follows.

$$\mathcal{M}(x, f; (r_x, r_f)) = \begin{cases} c_x \leftarrow \mathsf{Enc}^{r_x}(\mathsf{msk}, x) \\ sk_{\hat{f}} \leftarrow \mathsf{KeyGen}^{r_f}(\mathsf{msk}, \hat{f}) \\ s \leftarrow \mathsf{Dec}(c_x, sk_{\hat{f}}) \end{cases} \tag{2}$$

for $(r_x, r_f) \leftarrow \mathcal{R}$.

From all that, we can now proceed to our main result of this section.

Theorem 1. *Let $\mathcal{F}$ be a family of functions, and let $\mathcal{M}'$ be a DP mechanism for functions in $\mathcal{F}$. Let* RFE *be a randomized functional encryption scheme for*

the family of randomized function $\hat{\mathcal{F}}$ derived from $\mathcal{F}$ as defined above. Let $\kappa \in \mathbb{N}$ be a security parameter. If RFE *is 1-*SEL-SIM *secure and $\mathcal{M}'$ is $(\epsilon_\kappa, \delta_\kappa)$-*DP *for Q queries and some δ_κ negligible over κ, then the DP mechanism over encrypted database $\mathcal{M}$ defined above is (Q, ϵ_κ)-*IND-CDP.

Proof. First note that as explained in Section 2.2, statistical differential privacy implies computational differential privacy, as such, we only need to prove the result for $\mathcal{M}'$ being (Q, ϵ_κ)-IND-CDP. We will prove this through a series of Games. Let $\mathcal{A}$ be a PPT adversary playing the $(Q, \cdot)$-IND-CDP attack game for mechanism $\mathcal{M}$, let $\mathsf{Sim} = (\mathsf{EncSim}, \mathsf{KeyGenSim})$ be the simulator algorithms for the RFE scheme and let $\kappa \in \mathbb{N}$ be a security parameter. Changes on Game i are made over Game $i - 1$.

Game 0. This is the attack game for $(Q, \cdot)$-IND-CDP and mechanism $\mathcal{M}$ as seen in its definition, but we will refer to the challenger as $\mathcal{B}$.

Game 1. In this game we change the value obtained from the decryption algorithm for an evaluation of the randomized function. As such, for the oracle $\mathcal{O}$ when receiving f^i, $\mathcal{B}$ samples $r_i \leftarrow \mathcal{R}$ and computes $s^i = f^i(x_b) + e(r_i)$. Finally, it sends to $\mathcal{A}$ the following response

$$\mathcal{O}^1(x_b, f^i, r_i) = \begin{cases} c_{x_b} \\ sk_{\hat{f}^i} \\ s^i. \end{cases}$$

Game 2. In this game we simulate the ciphertext and functional keys using the RFE scheme simulator. As such, after choosing the bit b, and receiving x_0, x_1 from $\mathcal{A}$, the challenger $\mathcal{B}$ uses $\mathsf{EncSim}(1^\kappa, \hat{\mathcal{F}})$ and obtains the simulated ciphertext $c^*_{x_b}$ and the state st'. It then simulates the the oracle as follows. When receiving the function f^i, $\mathcal{B}$ executes the algorithm $\mathsf{KeyGenSim}(\mathsf{st}', \hat{f}^i)$ substituting the call to KeyIdeal for s^i and receives the simulated functional key $sk^*_{\hat{f}^i}$. Finally, it sends to $\mathcal{A}$ the following response

$$\mathcal{O}^2(x_b, f^i, r_i) = \begin{cases} c^*_{x_b} \\ sk^*_{\hat{f}^i} \\ s^i. \end{cases}$$

Game 3. In this game, $\mathcal{B}$ will act as a challenger for the adversary $\mathcal{A}$ in the $(Q, \cdot)$-IND-CDP attack game for mechanism $\mathcal{M}$ while acting as an adversary for the challenger $\mathcal{C}$ in the $(Q, \cdot)$-IND-CDP attack game for mechanism $\mathcal{M}'$. More concretely, when $\mathcal{B}$ receives x_0, x_1 from $\mathcal{A}$ it forwards them to $\mathcal{C}$. Then, for challenge queries, when receiving function f^i, $\mathcal{B}$ queries oracle $\mathcal{O}'$ to $\mathcal{C}$ and receives $v^i = \mathcal{M}'(x_b, f^i, r^i)$ for some $r_i \leftarrow \mathcal{R}$. It substitutes s^i in Game 1 for v^i and sends

to $\mathcal{A}$ the following response.

$$\mathcal{O}^3(x_b, f^i, r_i) = \begin{cases} c^*_{x_b} \\ sk^*_{\hat{f}^i} \\ v^i. \end{cases}$$

Analysis. Let $\mathcal{C}'$ be a challenger that chooses $b \in \{0,1\}$ uniformly at random. If $b = 0$ it interacts with the adversary $\mathcal{A}$ as in Game i, otherwise it interacts as in Game j. At the end of the interaction, $\mathcal{A}$ will make its guess $\tilde{b} \in \{0,1\}$. We define

$$\mathsf{Adv}_{ij}(\mathcal{A}) := \left| \Pr\left[\tilde{b} = b\right] - \frac{1}{2} \right|$$

for $i = 0, 1$ and $j = i + 1$. Also, since we know $\mathsf{Adv}_{ij}(\mathcal{A})$ is the advantage on distinguishing Game i from Game j, for any function over a Game g, i.e. any computable value from information from the game, the following holds

$$|g(\text{Game } i) - g(\text{Game } j)| \leq \mathsf{Adv}_{ij}(\mathcal{A}).$$

For this proof, let us define for any Game i the function

$$\mathsf{Adv}^i_{\mathsf{DP}} = \frac{\Pr^i\left[\tilde{b} = 1 | b = 0\right]}{\Pr^i\left[\tilde{b} = 1 | b = 1\right]}.$$

From Game 0 to Game 1. It is clear that $\mathsf{Dec}(c_{x_b}, sk_{f^i})$ and $f^i(x_b) + e(r_i)$ are computationally indistinguishable because of the correctness of the RFE scheme, which means that $\mathsf{Adv}_{01}(\mathcal{A}) = 0$ and therefore

$$\mathsf{Adv}^0_{\mathsf{DP}}(\mathcal{A}) = \mathsf{Adv}^1_{\mathsf{DP}}(\mathcal{A}).$$

From Game 1 to Game 2. The only change is swapping all the non-simulated algorithms from RFE for their simulators. It is clear that if $\mathcal{A}$ could distinguish between Game 1 and Game 2 we could construct an adversary $\mathcal{A}'$ able to distinguish the real and ideal experiments for the 1-SEL-SIM security game for RFE. As such we get that

$$\mathsf{Adv}_{12}(\mathcal{A}) \leq \mathsf{Adv}_{\text{1-SEL-SIM-RFE}}(\mathcal{A}') \leq \epsilon_{\text{1-SEL-SIM-RFE}}$$

and therefore

$$\left|\mathsf{Adv}^1_{\mathsf{DP}}(\mathcal{A}) - \mathsf{Adv}^2_{\mathsf{DP}}(\mathcal{A})\right| \leq \epsilon_{\text{1-SEL-SIM-RFE}}.$$

From Game 2 to Game 2. The view for adversary $\mathcal{A}$ does not change, since s^i and v^i are identically distributed by definition. This means that $\mathsf{Adv}_{23}(\mathcal{A}) = 0$ and therefore

$$\mathsf{Adv}^2_{\mathsf{DP}}(\mathcal{A}) = \mathsf{Adv}^3_{\mathsf{DP}}(\mathcal{A})$$

and in turn, given that $\mathcal{B}$ (as an adversary to $\mathcal{C}$) has the same output as $\mathcal{A}$ we get

$$\mathsf{Adv}^3_{\mathsf{DP}}(\mathcal{A}) = \mathsf{Adv}^3_{\mathsf{DP}}(\mathcal{B}).$$

To conclude the proof, we assume that $\mathcal{M}'$ is (Q, ϵ_κ)-IND-CDP, so for adversary $\mathcal{B}$ we know

$$\Pr^3\left[\tilde{b} = 1 | b = 0\right] \leq e^{\epsilon_\kappa} \cdot \Pr^3\left[\tilde{b} = 1 | b = 1\right] + \mathrm{negl}^2(\kappa) \tag{3}$$

for some negligible function $\mathrm{negl}^2(\cdot)$, which implies

$$\mathsf{Adv}^3_{\mathsf{DP}}(\mathcal{B}) \leq e^{\epsilon_\kappa} + \frac{\mathrm{negl}^2(\kappa)}{\Pr^3\left[\tilde{b} = 1 | b = 1\right]}.$$

Note that since $\Pr^3\left[\tilde{b} = 1 | b = 0\right] + \Pr^3\left[\tilde{b} = 1 | b = 1\right] = 1$ and inequality 3 holds, then $\Pr^3\left[\tilde{b} = 1 | b = 1\right]$ cannot be negligible, which in turn means that

$$\frac{\mathrm{negl}^2(\kappa)}{\Pr^3\left[\tilde{b} = 1 | b = 1\right]} = \mathrm{negl}^1(\kappa)$$

for some negligible function $\mathrm{negl}^1(\cdot)$.

Finally, using the results from the transitions between games we get

$$\begin{aligned}\mathsf{Adv}^0_{\mathsf{DP}}(\mathcal{A}) &\leq \mathsf{Adv}^3_{\mathsf{DP}}(\mathcal{B}) + \epsilon_{1-\mathsf{SIM}-\mathsf{FE}} \\ &\leq e^{\epsilon_\kappa} + \mathrm{negl}^1(\kappa) + \epsilon_{1-\mathsf{SIM}-\mathsf{FE}}\end{aligned}$$

which implies

$$\begin{aligned}\Pr^0\left[\tilde{b} = 1 | b = 0\right] &\leq e^{\epsilon_\kappa} \cdot \Pr^0\left[\tilde{b} = 1 | b = 1\right] + \\ &\quad + (\mathrm{negl}^1(\kappa) + \epsilon_{1-\mathsf{SIM}-\mathsf{FE}}) \cdot \Pr^0\left[\tilde{b} = 1 | b = 1\right] \\ &\leq e^{\epsilon_\kappa} \cdot \Pr^0\left[\tilde{b} = 1 | b = 1\right] + \mathrm{negl}^0(\kappa)\end{aligned}$$

for some negligible function $\mathrm{negl}^0(\cdot)$, as we wanted to see.

Remark 1. Theoretically, we can obtain an RFE scheme from a FE scheme in a generic manner [7]. However, this approach results in inefficient constructions. In the following section, we show that an efficient construction can be achieved for the class of inner-product functions.

4 Randomized Inner-Product Scheme

In this section we present our instantiation of a randomized inner-product functional encrypion scheme using an arbitrary IPFE scheme and prove its security. We finally obtain a concrete RIPFE scheme, from the general result of Theorem 1 in the previous section.

The most important concept to take a hold of is the fact that the noise must be sampled during the key generation phase since it must be different for every query while at the same time it must be hidden to satisfy differential privacy. The naive idea is then to use function-hiding inner product functional encryption [13] to hide the noise. However, it would need to be used as a building block towards constructing a RIPFE scheme, and would only result in a pairing-based scheme (very expensive with respect to exponentiations), since achieving function-hiding for inner products without pairings is a well-known open problem. To circumvent that, we expand on the ideas by Hamdi in [28], using the concept and the construction of a multi-input functional encryption scheme for inner product introduced in [4]. More precisely, the function we want to implement is seen as a two-input function:

- one is the message $\boldsymbol{x}$ is used during the encryption phase with a long-term key $\boldsymbol{u}$, as $\boldsymbol{d} = \boldsymbol{x} + \boldsymbol{u}$. To manage the fact that a ciphertext can be used several times with several different functional key queries, we then encrypt such one-time ciphertext using a standard IPFE ; and
- one is a DP noise $e_{\boldsymbol{y}}$ is used during the key generation phase with an ephemeral key $u'_{\boldsymbol{y}}$, as $d'_{\boldsymbol{y}} \leftarrow e_{\boldsymbol{y}} + u'_{\boldsymbol{y}}$.

Next, our functional key generation generates (i) one functional secret for the vector $\boldsymbol{y}$ using the master secret key of the basic IPFE, and (ii) one functional key related to the two-input function encryption of [4], as $zk_{\boldsymbol{y}} \leftarrow \langle \boldsymbol{u}, \boldsymbol{y} \rangle + u'_{\boldsymbol{y}}$. Finally, using the IPFE decryption and the property of the two-input functional encryption, we can easily recover $\langle \boldsymbol{x}, \boldsymbol{y} \rangle + e_{\boldsymbol{y}}$.

The final detail we must be careful with is the use of one-time pads and which finite group we are using them in. To be able to apply exactly our previous description, the base IPFE scheme would need to take inputs from a finite group, and have its outputs on the *exact same* finite group so as to be able to subtract the one-time pads out. There exist some instantiations that satisfy this, for example the ones in sections 4.2 and 5.2 in [9]. However, most efficient instantiations take bounded inputs inside $\mathbb{Z}$ which makes the use of the one-time pad non-trivial to implement. Despite that, by using a property which most of current instantiations of IPFE satisfy called two-step decryption (first defined in [4]). The basic idea is that the bounded integer inputs are encoded into finite group where the operations of the scheme are performed and then the results are decoded back into $\mathbb{Z}$. It is in this intermediate finite group where the one-time pad is performed.

4.1 Formal Description of the Scheme

Now we can give the formal description. Let $\ell, X, Y \in \mathbb{Z}_{>0}$ and $\mathcal{F}^{\ell,X,Y}$ be the family of inner products such that $\boldsymbol{y} \in \mathcal{F}^{\ell,X,Y}$ means that $\boldsymbol{y}(\boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$ for any $\boldsymbol{x} \in \mathbb{Z}^\ell$ with $\|\boldsymbol{x}\|_\infty < X$ and $\boldsymbol{y} \in \mathbb{Z}^\ell$ with $\|\boldsymbol{y}\|_\infty < Y$. Let $\mathsf{IPFE} = (\mathsf{SetUp}^{\mathsf{IPFE}}, \mathsf{Enc}^{\mathsf{IPFE}}, \mathsf{KeyGen}^{\mathsf{IPFE}}, \mathsf{Dec}^{\mathsf{IPFE}})$ be an inner-product functional encryption scheme for the family of functions $\mathcal{F}^{\ell,X,Y}$ that satisfies the following property: two-step decryption.

Property 1 (Adapted from Property 1, [4]). An inner-product functional encryption scheme $\mathsf{IPFE} = (\mathsf{SetUp}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ satisfies the *two-step decryption* property if there exist PPT algorithms $\mathsf{SetUp}', \mathsf{Dec1}, \mathsf{Dec2}$ and a function $\mathcal{E}$ such that:

1. For all $\kappa, \ell, X, Y \in \mathbb{Z}_{>0}$, the algorithm $\mathsf{SetUp}'(1^\kappa, \mathcal{F}^{\ell,X,Y})$ outputs $(\mathsf{param}, \mathsf{msk})$ where param contains a bound $B \in \mathbb{Z}_{>0}$ and a description of a commutative group $\mathbb{G}$ (with operation $\circ$) of order $L > \ell \cdot X \cdot Y$, defining the function $\mathcal{E} : \mathbb{Z}_L \times \mathbb{Z} \to \mathbb{G}$.
2. For all $(\mathsf{param}, \mathsf{msk}) \leftarrow \mathsf{SetUp}'(1^\kappa, \mathcal{F}^{\ell,X,Y})$, $c_{\boldsymbol{x}} \leftarrow \mathsf{Enc}(\mathsf{msk}, \boldsymbol{x})$ and $sk_{\boldsymbol{y}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \boldsymbol{y})$ we have
$$\mathsf{Dec1}(c_{\boldsymbol{x}}, sk_{\boldsymbol{y}}) = \mathcal{E}(\langle \boldsymbol{x}, \boldsymbol{y} \rangle, \mathsf{noise}(c_{\boldsymbol{x}}, sk_{\boldsymbol{y}}))$$
for some noise function. Furthermore, it holds for all $\boldsymbol{x}, \boldsymbol{y}$, $\Pr[\mathsf{noise}(c_{\boldsymbol{x}}, sk_{\boldsymbol{y}}) > B] < \mathsf{negl}(\kappa)$. Note that we are assuming that the encryption algorithm works for inputs greater than the bound.
3. Given any $\gamma \in \mathbb{Z}_L$ and param, $\mathcal{E}(\gamma, 0)$ can be efficiently computed.
4. The function $\mathcal{E}$ is linear, more specifically for any $\gamma, \gamma' \in \mathbb{Z}_L$ and any $\mathsf{noise}, \mathsf{noise}' \in \mathbb{Z}$, we have
$$\mathcal{E}(\gamma, \mathsf{noise}) \circ \mathcal{E}(\gamma', \mathsf{noise}') = \mathcal{E}(\gamma + \gamma', \mathsf{noise} + \mathsf{noise}').$$
5. For all $\gamma < \ell \cdot X \cdot Y$, and $\mathsf{noise} < \ell \cdot B$, $\mathsf{Dec2}(\mathcal{E}(\gamma, \mathsf{noise})) = \gamma$.

In other words, the decryption is done in two steps, where only the second one is affected by the bound on the inputs and its inverse can be computed efficiently only knowing the public parameters. The basic example are schemes based on the DDH assumption (Section 3 in [8]), where the function $\mathcal{E}(\gamma, \mathsf{noise}) = g^\gamma$ with g being the generator of the cyclic group $\mathbb{G}$ stated in the public parameters. It is proven in [4] that LWE and DCR based constructions also satisfy this property (for example Section 4 in [9] and Section 4 in [8]). It is for the inclusion of instantiations based on approximate encryption like LWE that the noise is incorporated to the function $\mathcal{E}$.

With this property defined, we proceed to describe our randomized scheme. Let D_ϵ be a probability distribution over $\mathbb{Z}$ and $\hat{\mathcal{F}}_\epsilon^{\ell,X,Y}$ as defined in Section 3. Then we define our randomized inner product functional encryption scheme $\mathsf{RIPFE} = (\mathsf{SetUp}^{\mathsf{RIPFE}}, \mathsf{Enc}^{\mathsf{RIPFE}}, \mathsf{KeyGen}^{\mathsf{RIPFE}}, \mathsf{Dec}^{\mathsf{RIPFE}})$ for the family of functions $\hat{\mathcal{F}}_\epsilon^{\ell,X,Y}$ as presented in Figure 2.

$\mathsf{SetUp}^{\mathsf{RIPFE}}(1^\kappa, \hat{\mathcal{F}}_\epsilon^{\ell,X,Y})$:
Choose distribution D_ϵ over $\mathbb{Z}$
Choose α such that $\Pr[|D_\epsilon| \geq \alpha] = \mathsf{negl}(\kappa)$
Choose $L > \ell \cdot X \cdot Y + \alpha$
$\boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_L^\ell$
$(\mathsf{param}^{\mathsf{IPFE}}, \mathsf{msk}^{\mathsf{IPFE}}) \leftarrow \mathsf{SetUp}^{\mathsf{IPFE}}(1^\kappa, \mathcal{F}^{\ell, X+\alpha/(\ell\cdot Y), Y})$
Output $(\mathsf{param}^{\mathsf{RIPFE}}, \mathsf{msk}^{\mathsf{RIPFE}}) = ((L, D_\epsilon, \mathsf{param}^{\mathsf{IPFE}}), (\boldsymbol{u}, \mathsf{msk}^{\mathsf{IPFE}}))$

$\mathsf{Enc}^{\mathsf{RIPFE}}(\mathsf{msk}^{\mathsf{RIPFE}}, \boldsymbol{x})$:
$\boldsymbol{d} \leftarrow \boldsymbol{x} + \boldsymbol{u} \pmod L$
$c_{\boldsymbol{d}} \leftarrow \mathsf{Enc}^{\mathsf{IPFE}}(\mathsf{msk}^{\mathsf{IPFE}}, \boldsymbol{d})$
Output $c_{\boldsymbol{d}}$

$\mathsf{KeyGen}^{\mathsf{RIPFE}}(\mathsf{msk}^{\mathsf{RIPFE}}, \boldsymbol{y})$:
$e_{\boldsymbol{y}} \leftarrow D_\epsilon$
$u'_{\boldsymbol{y}} \xleftarrow{\$} \mathbb{Z}_L$
$d'_{\boldsymbol{y}} \leftarrow e_{\boldsymbol{y}} + u'_{\boldsymbol{y}} \pmod L$
$sk_{\boldsymbol{y}} \leftarrow \mathsf{KeyGen}^{\mathsf{IPFE}}(\mathsf{msk}^{\mathsf{IPFE}}, \boldsymbol{y})$
$zk_{\boldsymbol{y}} \leftarrow \langle \boldsymbol{u}, \boldsymbol{y} \rangle + u'_{\boldsymbol{y}} \pmod L$
Output $sk_{\boldsymbol{y}}^{\mathsf{RIPFE}} = (d'_{\boldsymbol{y}}, sk_{\boldsymbol{y}}, zk_{\boldsymbol{y}})$

$\mathsf{Dec}^{\mathsf{RIPFE}}(c_d, sk_y^{\mathsf{RIPFE}})$:
$\mathcal{E}(\langle \boldsymbol{d}, \boldsymbol{y} \rangle, \mathsf{noise}(c_{\boldsymbol{x}}, sk_{\boldsymbol{y}})) \leftarrow \mathsf{Dec1}^{\mathsf{IPFE}}(c_{\boldsymbol{d}}, sk_{\boldsymbol{y}})$
$s \leftarrow \mathsf{Dec2}(\mathcal{E}(\langle \boldsymbol{d}, \boldsymbol{y} \rangle, \mathsf{noise}(c_{\boldsymbol{x}}, sk_{\boldsymbol{y}})) \circ \mathcal{E}(d'_{\boldsymbol{y}} - zk_{\boldsymbol{y}}, 0))$
Output s

Fig. 2. Randomized inner-product functional encryption scheme RIPFE.

4.2 Correctness and Security

First we need to verify that this is a correct randomized functional encryption scheme for $\hat{\mathcal{F}}_\epsilon^{\ell,X,Y}$.

Proposition 1. *The* RIPFE *scheme defined in Figure 2 is a correct randomized functional encryption scheme for* $\hat{\mathcal{F}}_\epsilon^{\ell,X,Y}$.

Proof. Let $\boldsymbol{x} \in \mathbb{Z}^\ell$ with $\|\boldsymbol{x}\|_\infty$ be any plaintext and $\hat{\boldsymbol{y}}^1, \ldots, \hat{\boldsymbol{y}}^Q \in \hat{\mathcal{F}}_\epsilon^{\ell,X,Y}$ any set of randomized functions. Then for any $i \in [Q]$ we get that $s^i \leftarrow \mathsf{Dec}(c_{\boldsymbol{d}}, sk_{\boldsymbol{y}_i}^{\mathsf{RIPFE}})$ satisfies the following.

$$\begin{aligned} s^i &= \langle \boldsymbol{d}, \boldsymbol{y}^i \rangle + d'_{\boldsymbol{y}^i} - zk_{\boldsymbol{y}^i} \\ &= \langle \boldsymbol{x}, \boldsymbol{y}^i \rangle + \langle \boldsymbol{u}, \boldsymbol{y}^i \rangle + e_{\boldsymbol{y}} + u'_{\boldsymbol{y}^i} - (\langle \boldsymbol{u}, \boldsymbol{y}^i \rangle + u'_{\boldsymbol{y}^i}) \\ &= \langle \boldsymbol{x}, \boldsymbol{y}^i \rangle + e_{\boldsymbol{y}^i}. \end{aligned}$$

This is clearly the same distribution as $\hat{\boldsymbol{y}}(\boldsymbol{x}; r)$ for $r \leftarrow \mathcal{R}$ since $e_{\boldsymbol{y}}$ is sampled from D_ϵ independently for every query $\boldsymbol{y}$.

We will now prove the simulation soundness of our scheme by lifting the security guarantee form the base IPFE scheme to the randomized version.

Theorem 2. *Let* IPFE *be a 1-*SEL-SIM*-secure inner product functional encryption scheme. Our construction* RIPFE *in Figure 2 is a 1-*SEL-SIM*-secure randomized functional encryption scheme.*

Proof. We will prove the result through a series of Games. Let $\mathcal{A}$ be a PPT adversary playing the 1-SEL-SIM security game for RIPFE, and let $\kappa \in \mathbb{N}$ be a security parameter. Changes on Game i are made over Game $i-1$. Let also $\mathsf{Sim}^{\mathsf{IPFE}} = (\mathsf{EncSim}^{\mathsf{IPFE}}, \mathsf{KeyGenSim}^{\mathsf{IPFE}})$ be the simulator for the IPFE scheme.

Game 0. This is the 1-SEL-SIM security real experiment for RIPFE as described below

$\mathsf{Exp}^0_{\mathcal{A}}(1^\kappa, \hat{\mathcal{F}}^{\ell,X,Y}_\epsilon)$
1: $(\boldsymbol{x}, \mathsf{st}_1) \leftarrow \mathcal{A}_1$ where $\boldsymbol{x} \in \mathbb{Z}^\ell$, $\|\boldsymbol{x}\|_\infty < X$
2: $(\mathsf{param}, \mathsf{msk}) \leftarrow \mathsf{SetUp}^0(1^\kappa, \hat{\mathcal{F}}^{\ell,X,Y}_\epsilon)$
3: $c_{\boldsymbol{d}} \leftarrow \mathsf{Enc}^0(\boldsymbol{x}, \mathsf{msk})$
4: $\gamma \leftarrow \mathcal{A}_2^{\mathcal{O}^0_1(\cdot), \mathcal{O}^0_2(\cdot,\cdot)}(c_{\boldsymbol{d}}, \mathsf{st}_1)$
Output: $(\boldsymbol{x}, \{\hat{\boldsymbol{y}}\}, \{sk_{\hat{\boldsymbol{y}}}\}\{s\}, \gamma)$

where SetUp^0, Enc^0, $\mathcal{O}^0_1$, $\mathcal{O}^0_2$ are the regular RIPFE algorithms and oracles.

Game 1. In this game we simulate the ciphertext. The experiment develops into the following.

$\mathsf{Exp}^1_{\mathcal{A}}(1^\kappa, \hat{\mathcal{F}}^{\ell,X,Y}_\epsilon)$
1: $(\boldsymbol{x}, \mathsf{st}_1) \leftarrow \mathcal{A}_1$ where $\boldsymbol{x} \in \mathbb{Z}^\ell$, $\|\boldsymbol{x}\|_\infty < X$
2: $(L, D_\epsilon, \mathsf{param}^{\mathsf{IPFE}}, \mathsf{msk}^{\mathsf{IPFE}}) \leftarrow \mathsf{SetUp}^1(1^\kappa, \hat{\mathcal{F}}^{\ell,X,Y}_\epsilon)$
3: $(c^*_{\boldsymbol{d}}, \mathsf{st}') \leftarrow \mathsf{Enc}^1(\mathsf{msk}^{\mathsf{IPFE}})$
4: $\gamma \leftarrow \mathcal{A}_2^{\mathcal{O}^1_1(\mathsf{msk}, \mathsf{st}', \boldsymbol{x}, \cdot), \mathcal{O}^1_2(\cdot,\cdot)}(c^*_{\boldsymbol{d}}, \mathsf{st}_1)$
Output: $(\boldsymbol{x}, \{\hat{\boldsymbol{y}}\}, \{sk_{\hat{\boldsymbol{y}}}\}\{s\}, \gamma)$

Where the Enc^1 and $\mathcal{O}^1_1$ algorithms are described below and SetUp^1, $\mathcal{O}^1_2$ are the same as in Game 0.

$\mathsf{SetUp}^1(1^\kappa, \hat{\mathcal{F}}_\epsilon^{\ell,X,Y})$:
Choose distribution D_ϵ over $\mathbb{Z}$
Choose α such that $\Pr[|D_\epsilon| \geq \alpha] = \mathsf{negl}(\kappa)$
Choose $L > \ell \cdot X \cdot Y + \alpha$
$(\mathsf{param}^{\mathsf{IPFE}}, \mathsf{msk}^{\mathsf{IPFE}}) \leftarrow \mathsf{SetUp}^{\mathsf{IPFE}}(1^\kappa, 1^\ell, X + \alpha/(\ell \cdot Y), Y)$
Output $(L, D_\epsilon, \mathsf{param}^{\mathsf{IPFE}}, \mathsf{msk}^{\mathsf{IPFE}})$

$\mathsf{Enc}^1(\mathsf{msk})$:
$\boldsymbol{d}^* \xleftarrow{\$} \mathbb{Z}_L^\ell$
$\mathsf{st}' \leftarrow \boldsymbol{d}^*$
$c_{\boldsymbol{d}^*} \leftarrow \mathsf{Enc}^{\mathsf{IPFE}}(\mathsf{msk}^{\mathsf{IPFE}}, \boldsymbol{d}^*)$
Output $(c_{\boldsymbol{d}^*}, \mathsf{st}')$

$\mathcal{O}_1^1(\mathsf{msk}, \mathsf{st}', \boldsymbol{x}, \hat{\boldsymbol{y}})$:
$e_{\boldsymbol{y}} \leftarrow D_\epsilon$
$u'_{\boldsymbol{y}} \xleftarrow{\$} \mathbb{Z}_L$
$d'_{\boldsymbol{y}} \leftarrow e_{\boldsymbol{y}} + u'_{\boldsymbol{y}} \pmod L$
$sk_{\boldsymbol{y}} \leftarrow \mathsf{KeyGen}^{\mathsf{IPFE}}(\mathsf{msk}^{\mathsf{IPFE}}, \boldsymbol{y})$
$zk^*_{\boldsymbol{y}} \leftarrow \langle \boldsymbol{d}^*, \boldsymbol{y} \rangle + u'_{\boldsymbol{y}} - \langle \boldsymbol{x}, \boldsymbol{y} \rangle \pmod L$
Output $sk^1_{\hat{\boldsymbol{y}}} = (d'_{\boldsymbol{y}}, sk_{\boldsymbol{y}}, zk^*_{\boldsymbol{y}})$

Game 2. In this game we will simulate the noise using the $\mathsf{KeyIdeal}$ functionality described in Definition 8. The experiment changes into the following.

$\mathsf{Exp}^2_{\mathcal{A}}(1^\kappa, \hat{\mathcal{F}}_\epsilon^{\ell,X,Y})$
1: $(\boldsymbol{x}, \mathsf{st}_1) \leftarrow \mathcal{A}_1$ where $\boldsymbol{x} \in \mathbb{Z}^\ell$, $\|\boldsymbol{x}\|_\infty < X$
2: $(L, D_\epsilon, \mathsf{param}^{\mathsf{IPFE}}, \mathsf{msk}^{\mathsf{IPFE}}) \leftarrow \mathsf{SetUp}^2(1^\kappa, \hat{\mathcal{F}}_\epsilon^{\ell,X,Y})$
3: $(c^*_{\boldsymbol{d}}, \mathsf{st}') \leftarrow \mathsf{Enc}^2(\mathsf{msk}^{\mathsf{IPFE}})$
4: $\gamma \leftarrow \mathcal{A}_2^{\mathcal{O}_1^2(\mathsf{msk}, \mathsf{st}', \cdot), \mathcal{O}_2^2(\cdot,\cdot)}(c^*_{\boldsymbol{d}}, \mathsf{st}_1)$
Output: $(\boldsymbol{x}, \{\hat{\boldsymbol{y}}\}, \{sk_{\hat{\boldsymbol{y}}}\}\{s\}, \gamma)$

Where the $\mathcal{O}_1^2$ algorithm is described below and SetUp^2, Enc^2 and $\mathcal{O}_2^2$ are the same as in Game 1.

$\mathcal{O}_1^2(\mathsf{msk}, \mathsf{st}', \hat{\boldsymbol{y}})$:
$d'^*_{\boldsymbol{y}} \xleftarrow{\$} \mathbb{Z}_L$
$v \leftarrow \mathsf{KeyIdeal}(\boldsymbol{x}, \hat{\boldsymbol{y}})$
$sk_{\boldsymbol{y}} \leftarrow \mathsf{KeyGen}^{\mathsf{IPFE}}(\mathsf{msk}^{\mathsf{IPFE}}, \boldsymbol{y})$
$zk^*_{\boldsymbol{y}} \leftarrow \langle \boldsymbol{d}^*, \boldsymbol{y} \rangle + d'^*_{\boldsymbol{y}} - v$
Output $sk^2_{\hat{\boldsymbol{y}}} = (d'^*_{\boldsymbol{y}}, sk_{\boldsymbol{y}}, zk^*_{\boldsymbol{y}})$

Game 3. In this Game we add a challenger $\mathcal{C}^*$ playing the 1-$\mathsf{SEL\text{-}SIM}$ security game for the IPFE scheme in the real world, with the added change that in the decryption oracle, were it to fail to compute the response due to the inputs being outside the bounds, it returns $\mathsf{Dec1}(c_{\boldsymbol{x}}, sk_{\boldsymbol{y}})$ instead of nothing. As such, the experiment changes into the following.

$\mathsf{Exp}^3_{\mathcal{A}}(1^\kappa, \hat{\mathcal{F}}^{\ell,X,Y}_\epsilon)$

1: $(\boldsymbol{x}, \mathsf{st}_1) \leftarrow \mathcal{A}_1$ where $\boldsymbol{x} \in \mathbb{Z}^\ell$, $\|\boldsymbol{x}\|_\infty < X$
2: $(L, D_\epsilon, \mathsf{param}^{\mathsf{IPFE}}, c^*_{\boldsymbol{d}}, \mathsf{st}') \leftarrow \mathsf{Enc}^{3,\mathcal{C}^*}(1^\kappa, \hat{\mathcal{F}}^{\ell,X,Y}_\epsilon)$
4: $\gamma \leftarrow \mathcal{A}_2^{\mathcal{O}_1^{3,\mathcal{C}^*}(\mathsf{st}',\cdot),\mathcal{O}_2^{3,\mathcal{C}^*}(\cdot,\cdot)}(c^*_{\boldsymbol{d}}, \mathsf{st}_1)$
Output: $(\boldsymbol{x}, \{\hat{\boldsymbol{y}}\}, \{sk_{\hat{\boldsymbol{y}}}\}\{s\}, \gamma)$

Where the Enc^3, $\mathcal{O}_1^{3,\mathcal{C}^*}$ and $\mathcal{O}_2^{3,\mathcal{C}^*}$ algorithms are described below.

$\mathsf{Enc}^{3,\mathcal{C}^*}(1^\kappa, \hat{\mathcal{F}}^{\ell,X,Y}_\epsilon)$:
Choose distribution D_ϵ over $\mathbb{Z}$
Choose α such that $\Pr[|D_\epsilon| \geq \alpha] = \mathsf{negl}(\kappa)$
Choose $L > \ell \cdot X \cdot Y + \alpha$
$\boldsymbol{d}^* \xleftarrow{\$} \mathbb{Z}_L^\ell$
$\mathsf{st}' \leftarrow \boldsymbol{d}^*$
$(\mathsf{param}^{\mathsf{IPFE}}, c_{\boldsymbol{d}^*}) \leftarrow \mathsf{Exp}^{real}_{\mathsf{IPFE}}(1^\kappa, \mathcal{F}^{\ell,X+\alpha/(\ell\cdot Y),Y}_\epsilon, \boldsymbol{d}^*)$
Output: $(L, D_\epsilon, \mathsf{param}^{\mathsf{IPFE}}, c_{\boldsymbol{d}^*}, \mathsf{st}')$

$\mathcal{O}_1^{3,\mathcal{C}^*}(\mathsf{st}', \boldsymbol{y})$:
$d'^*_{\boldsymbol{y}} \xleftarrow{\$} \mathbb{Z}_L$
$v \leftarrow \mathsf{KeyIdeal}(\boldsymbol{x}, \hat{\boldsymbol{y}})$
$zk^*_{\boldsymbol{y}} \leftarrow \langle \boldsymbol{d}^*, \boldsymbol{y} \rangle + d'^*_{\boldsymbol{y}} - v$
$sk_{\boldsymbol{y}} \leftarrow \mathcal{O}_1^{\mathcal{C}^*}(\boldsymbol{y})$
Output $sk^3_{\hat{\boldsymbol{y}}} = (d'^*_{\boldsymbol{y}}, sk_{\boldsymbol{y}}, zk^*_{\boldsymbol{y}})$

$\mathcal{O}_2^{3,\mathcal{C}^*}(c_{\boldsymbol{x}}, sk^3_{\boldsymbol{y}})$
If $\mathsf{Dec}^{\mathsf{IPFE}}$ works
  $s \leftarrow \mathcal{O}_2^{\mathcal{C}^*}(c_{\boldsymbol{x}}, sk_{\boldsymbol{y}})$
  Output: $s + d'^*_{\boldsymbol{y}} - zk^*_{\boldsymbol{y}}$
Else
  $s \leftarrow \mathcal{O}_2^{\mathcal{C}^*}(c_{\boldsymbol{x}}, sk_{\boldsymbol{y}})$
  Output: $\mathsf{Dec2}(s \circ \mathcal{E}(d'^*_{\boldsymbol{y}} - zk^*_{\boldsymbol{y}}, 0))$

Game 4. In this Game we add the simulators from the base IPFE. The experiment develops into the following.

$\mathsf{Exp}^4_{\mathcal{A}}(1^\kappa, \hat{\mathcal{F}}^{\ell,X,Y}_\epsilon)$

1: $(\boldsymbol{x}, \mathsf{st}_1) \leftarrow \mathcal{A}_1$ where $\boldsymbol{x} \in \mathbb{Z}^\ell$, $\|\boldsymbol{x}\|_\infty < X$
2: $(L, D_\epsilon, \mathsf{param}^{\mathsf{IPFE}}, c^*_{\boldsymbol{d}}, \mathsf{st}') \leftarrow \mathsf{Enc}^{4,\mathcal{C}^*}(1^\kappa, 1^\ell, 1^\epsilon, X, Y)$
4: $\gamma \leftarrow \mathcal{A}_2^{\mathcal{O}_1^{4,\mathcal{C}^*}(\mathsf{st}',\cdot),\mathcal{O}_2^{4,\mathcal{C}^*}(\cdot,\cdot)}(c^*_{\boldsymbol{d}}, \mathsf{st}_1)$
Output: $(\boldsymbol{x}, \{\hat{\boldsymbol{y}}\}, \{sk_{\hat{\boldsymbol{y}}}\}\{s\}, \gamma)$

Where the Enc^4, $\mathcal{O}_1^4$ and $\mathcal{O}_2^4$ algorithms are described below.

$\mathsf{Enc}^{4,\mathcal{C}^*}(1^\kappa, \hat{\mathcal{F}}_\epsilon^{\ell,X,Y})$:
Choose distribution D_ϵ over $\mathbb{Z}$
Choose α such that $\Pr[|D_\epsilon| \geq \alpha] = \mathsf{negl}(\kappa)$
Choose $L > \ell \cdot X \cdot Y + \alpha$
$\boldsymbol{d}^* \xleftarrow{\$} \mathbb{Z}_L^\ell$
$\mathsf{st}' \leftarrow \boldsymbol{d}^*$
$(\mathsf{param}^{\mathsf{IPFE}}, c_{\boldsymbol{d}^*}) \leftarrow \mathsf{Exp}^{ideal}_{\mathsf{IPFE}}(1^\kappa, \mathcal{F}_\epsilon^{\ell, X+\alpha/(\ell\cdot Y), Y}, \boldsymbol{d}^*)$
Output: $(L, D_\epsilon, \mathsf{param}^{\mathsf{IPFE}}, c_{\boldsymbol{d}^*}, \mathsf{st}')$

$\mathcal{O}_1^{4,\mathcal{C}^*}(\mathsf{st}', \boldsymbol{y})$:
$d'^*_{\boldsymbol{y}} \xleftarrow{\$} \mathbb{Z}_L$
$v \leftarrow \mathsf{KeyIdeal}(\boldsymbol{x}, \hat{\boldsymbol{y}})$
$zk^*_{\boldsymbol{y}} \leftarrow \langle \boldsymbol{d}^*, \boldsymbol{y}\rangle + d'^*_{\boldsymbol{y}} - v$
$sk_{\boldsymbol{y}} \leftarrow \mathcal{O}_1'^{\mathcal{C}^*}(\boldsymbol{y})$
Output $sk^3_{\hat{\boldsymbol{y}}} = (d'^*_{\boldsymbol{y}}, sk_{\boldsymbol{y}}, zk^*_{\boldsymbol{y}})$

$\mathcal{O}_2^{4,\mathcal{C}^*}(c_{\boldsymbol{x}}, sk^3_{\boldsymbol{y}})$
If $\mathsf{Dec}^{\mathsf{IPFE}}$ works
  $s \leftarrow \mathcal{O}_2'^{\mathcal{C}^*}(c_{\boldsymbol{x}}, sk_{\boldsymbol{y}})$
  Output: $s + d'^*_{\boldsymbol{y}} - zk^*_{\boldsymbol{y}}$
Else
  $s \leftarrow \mathcal{O}_2'^{\mathcal{C}^*}(c_{\boldsymbol{x}}, sk_{\boldsymbol{y}})$
  Output: $\mathsf{Dec2}(s \circ \mathcal{E}(d'^*_{\boldsymbol{y}} - zk^*_{\boldsymbol{y}}, 0))$

Game 5. In this game we finalize the simulation. As such, the experiment remains as follows.

$\mathsf{Exp}^5_{\mathcal{A}}(1^\kappa, \hat{\mathcal{F}}_\epsilon^{\ell,X,Y})$
1: $(\boldsymbol{x}, \mathsf{st}_1) \leftarrow \mathcal{A}_1$ where $\boldsymbol{x} \in \mathbb{Z}^\ell$, $\|\boldsymbol{x}\|_\infty < X$
2: $(c^*_{\boldsymbol{d}}, \mathsf{st}') \leftarrow \mathsf{EncSim}^{\mathcal{C}^*}(1^\kappa, \hat{\mathcal{F}}_\epsilon^{\ell,X,Y})$
3: $\gamma \leftarrow \mathcal{A}_2^{\mathcal{O}_1'^{\mathcal{C}^*}(\mathsf{st}',\cdot), \mathcal{O}_2'^{\mathcal{C}^*}(\cdot,\cdot)}(c^*_{\boldsymbol{d}}, \mathsf{st}_1)$
Output: $(\boldsymbol{x}, \{\hat{\boldsymbol{y}}'\}, \{sk_{\hat{\boldsymbol{y}}'}\}\{s'\}, \gamma)$

Where $\mathsf{EncSim}^{\mathcal{C}^*}$ corresponds to $\mathsf{Enc}^{4,\mathcal{C}^*}$, $\mathcal{O}_1'^{\mathcal{C}^*}(\mathsf{st}', \cdot)$ immediately calls $\mathcal{O}_1^{4,\mathcal{C}^*}(\mathsf{st}', \cdot)$, and $\mathcal{O}_2'^{\mathcal{C}^*}(\cdot,\cdot)$ immediately calls $\mathcal{O}_2^{4,\mathcal{C}^*}(\cdot,\cdot)$.

Analysis. Let $\mathcal{C}'$ be a challenger that chooses $b \in \{0,1\}$ uniformly at random. If $b = 0$ it interacts with the adversary $\mathcal{A}$ as in Game i, otherwise it interacts as in Game j. At the end of the interaction, $\mathcal{A}$ will make its guess $\tilde{b} \in \{0,1\}$. We define

$$\mathsf{Adv}_{i(i+1)}(\mathcal{A}) := \left| \Pr\left[\tilde{b} = b\right] - \frac{1}{2} \right|$$

for $i = 0, 1$.

From Game 0 to Game 1. In this change, we have swapped $\boldsymbol{u} + \boldsymbol{x}$ for $\boldsymbol{d}^*$ in encryption (since in key generation $\boldsymbol{u} = \boldsymbol{d}^* - \boldsymbol{x}$). As such, given that the secret key $\boldsymbol{u}$ and the challenge $\boldsymbol{x}$ are chosen independently, which means that $\boldsymbol{u} + \boldsymbol{x}$ and $\boldsymbol{d}^*$ are equally distributed. Therefore, $\mathsf{Adv}_{01}(\mathcal{A}) = 0$.

From Game 1 to Game 2. Analogously to the previous step, $u'_{\boldsymbol{y}}$ and $e_{\boldsymbol{y}}$ are independent, and therefore, both $u'_{\boldsymbol{y}} + e_{\boldsymbol{y}}$ and $d'^*_{\boldsymbol{y}}$ are equally distributed (and $u'_{\boldsymbol{y}} = d'^*_{\boldsymbol{y}} - e_{\boldsymbol{y}}$). Then, $\mathsf{Adv}_{12}(\mathcal{A}) = 0$.

From Game 2 to Game 3. Game 3 is a rewriting of Game 2 but using the real experiment for the base IPFE with challenger $\mathcal{C}^*$, where the view of the adversary is not modified in any way. Therefore $\mathsf{Adv}_{23}(\mathcal{A}) = 0$.

From Game 3 to Game 4. In this change we have swapped from the real to the ideal experiment in the base IPFE scheme. As such, the distinguishing game between Game 2 and Game 3 is the 1-SEL-SIM game for the IPFE scheme with challenger $\mathcal{C}^*$. Therefore, $\mathsf{Adv}_{34}(\mathcal{A}) \leq \epsilon^*_{1-\mathsf{SEL}-\mathsf{SIM}-\mathsf{IPFE}}$.

From Game 4 to Game 5. Game 5 is a rewriting of Game 4, where the view of the adversary is not modified in any way. Therefore $\mathsf{Adv}_{45}(\mathcal{A}) = 0$.

Finally, adding it all up and considering that Game 0 is the real experiment and Game 5 is the ideal experiment we get that

$$\begin{aligned}\mathsf{Adv}_{real/ideal}(\mathcal{A}) &= \mathsf{Adv}_{01}(\mathcal{A}) + \mathsf{Adv}_{12}(\mathcal{A}) + \mathsf{Adv}_{23}(\mathcal{A}) + \mathsf{Adv}_{34}(\mathcal{A}) + \mathsf{Adv}_{45}(\mathcal{A}) \\ &= \epsilon^*_{1-\mathsf{SEL}-\mathsf{SIM}-\mathsf{IPFE}}\end{aligned}$$

as we wanted to see.

Remark 2. Note that the simulation soundness in which we base our result is against a challenger for the IPFE scheme $\mathcal{C}^*$ who in case of failure of the decryption algorithm outputs $\mathsf{Dec1}^{\mathsf{IPFE}}(c_{\boldsymbol{d}}, sk_{\boldsymbol{y}})$ instead of returning nothing nothing which is the response for the challenger $\mathcal{C}$ in standard simulation soundness for IPFE schemes. However, we argue that given the fact that our security model does not contemplate a decryption oracle with inputs a ciphertext and a function (instead of ciphertext and functional key) both challengers are equivalent.

Remark 3. Note that even if we have only proven 1-SEL-SIM security for our proposal, this implies 1-SEL-IND which can be then extended to arbitrarily many ciphertexts, as shown in [27]. This means that, in terms of security, by relying on the less constraining indistinguishability security we have selective security for arbitrarily many ciphertexts.

4.3 Efficiency Considerations

For a generic IPFE, our RIPFE scheme needs no extra inner-product slot to handle the noise, in other words, it is constructed with little overcost (both in storage

Table 5. Generic efficiency estimates for RIPFE.

	msk	$c_{\boldsymbol{x}}$	$sk_{\boldsymbol{y}}$
Size	$\mathsf{IPFE}^{\ell}_{\mathsf{msk}} + \ell\lvert\mathbb{G}\rvert$	$\mathsf{IPFE}^{\ell}_{c_{\boldsymbol{x}}}$	$\mathsf{IPFE}^{\ell}_{sk_{\boldsymbol{y}}} + 2\lvert\mathbb{G}\rvert$

	SetUp	Enc	KeyGen	Dec
Comp. time	$\mathsf{IPFE}^{\ell}_{\mathsf{SetUp}} + \ell \cdot t_{Usampl}$	$\mathsf{IPFE}^{\ell}_{\mathsf{Enc}} + \ell \cdot t_{add}$	$\mathsf{IPFE}^{\ell}_{\mathsf{KeyGen}} + t_{Dsampl} + t_{Usampl} + \ell \cdot t_{prod} + (\ell+1) \cdot t_{add}$	$\mathsf{IPFE}^{\ell}_{\mathsf{Dec}} + t_{add} + t_{subs}$

as in computation time) in respect to the base IPFE scheme. More in detail, in storage this overcost consists in the extra one-time pad key $\boldsymbol{u}$ in the master secret key and two extra integers (namely $d'_{\boldsymbol{y}}$ and $zk_{\boldsymbol{y}}$) in the functional decryption key. In computation time the overcost consists: during the SetUp the sampling of $\boldsymbol{u}$, in Enc an extra ℓ additions, in KeyGen sampling $e_{\boldsymbol{y}}$ and $u'_{\boldsymbol{y}}$ together with an inner product and two extra additions and in Dec there is an extra addition and substraction as well as the computation of the function $\mathcal{E}$. All those elements are summarized in Table 5. In regards to the notation on this table, the integer κ is the security parameter, and we denote as IPFE^{ℓ}_{s} the size or computation time (depending on what the string s makes reference to) of the base IPFE scheme for ℓ coefficients.

5 Private encrypted database

In this section we describe our full system for a computationally differentially private encrypted database supporting linear queries, following the model given in Section 2.4. Our system is based on the randomized inner product functional encryption scheme given in the previous section, and the generic DP mechanism described in Section 3. We prove that such system is secure and private even against a collusion between the analyst and the server, using the security results of the two previous sections. We finally give some words about a practical deployment of such system.

5.1 Description of the System

Let ***DO*** be a Data Owner, ***S*** be an external server that stores sensitive databases, and let ***A*** be an analyst wanting to make requests on the stored databases. The overall idea of our system is to use the randomized inner product scheme to cover the private queries from the analyst, and to get advantage of the non-noisy IPFE scheme embeded into the randomized version (see the previous section for details) to answer the queries from the database owner, thanks to some non-noisy keys. Indeed, following the formalization given in Section 2.4, the **EQuery** from ***DO*** are only based on the IPFE, while the **PQuery** from ***A*** are based on the RIPFE. The latter is divided into two parts: the **SetUp** and the **KeyGen** are executed during the system setup and the **Dec** is done during the query part.

Let us now formally describe the private encrypted database. Let D_{ϵ} be a distribution over $\mathbb{Z}$, let $\mathsf{IPFE} = (\mathsf{SetUp}^{\mathsf{IPFE}}, \mathsf{Enc}^{\mathsf{IPFE}}, \mathsf{KeyGen}^{\mathsf{IPFE}}, \mathsf{Dec}^{\mathsf{IPFE}})$ be a

generic inner product functional encryption scheme for the family of functions $\mathcal{F}^{\ell,X,Y}$ satisfying two-step decryption and let $\mathsf{RIPFE} = (\mathsf{SetUp}^{\mathsf{RIPFE}}, \mathsf{Enc}^{\mathsf{RIPFE}}, \mathsf{KeyGen}^{\mathsf{RIPFE}}, \mathsf{Dec}^{\mathsf{RIPFE}})$ be the randomized inner product functional encryption scheme described in Figure 2, defined using the same IPFE. This is essentially that the same IPFE scheme is used both alone for the **EQuery** from $\boldsymbol{DO}$ and as the underlying primitive for the RIPFE scheme used for the **PQuery**.

Our private encrypted database supporting inner product queries over a static database $\mathsf{PIPFE} = (\mathsf{SetUp}, \mathsf{EQuery}, \mathsf{PQuery})$ is then given in Figure 3.

$\mathbf{SetUp}_{\boldsymbol{DO},\boldsymbol{S},\boldsymbol{A}}((1^\kappa, \hat{\mathcal{F}}_\epsilon^{\ell,X,Y}, \boldsymbol{x}); \perp; (\boldsymbol{y}^1 \dots \boldsymbol{y}^Q))$:
1. $\boldsymbol{DO}$ computes the following
   a. $(\mathsf{msk}^{\mathsf{RIPFE}}, param^{\mathsf{RIPFE}}) \leftarrow \mathsf{SetUp}^{\mathsf{RIPFE}}(1^\kappa, \hat{\mathcal{F}}_\epsilon^{\ell,X,Y})$.
   b. $c_{\boldsymbol{x}} \leftarrow \mathsf{Enc}^{\mathsf{RIPFE}}(\mathsf{msk}^{\mathsf{RIPFE}}, \boldsymbol{x})$.
   c. $sk_{\boldsymbol{y}^i}^{\mathsf{RIPFE}} \leftarrow \mathsf{KeyGen}^{\mathsf{RIPFE}}(\mathsf{msk}^{\mathsf{RIPFE}}, \boldsymbol{y}^i)$ for $i \in [Q]$.
2. $\boldsymbol{DO}$ keeps $\mathsf{msk}^{\mathsf{RIPFE}}$ secret.
3. $\boldsymbol{S}$ receives $c_{\boldsymbol{x}}$.
4. $\boldsymbol{A}$ receives $sk_{\boldsymbol{y}^1}^{\mathsf{RIPFE}}, \dots, sk_{\boldsymbol{y}^Q}^{\mathsf{RIPFE}}$

$\mathbf{EQuery}_{\boldsymbol{DO},\boldsymbol{S}}((\mathsf{msk}^{\mathsf{RIPFE}}, \boldsymbol{g}); c_{\boldsymbol{x}})$:
1. $\boldsymbol{DO}$ computes $sk_{\boldsymbol{g}}^{\mathsf{IPFE}} \leftarrow \mathsf{KeyGen}^{\mathsf{IPFE}}(\mathsf{msk}^{\mathsf{IPFE}}, \boldsymbol{g})$.
2. $\boldsymbol{DO}$ receives $c_{\boldsymbol{x}}$ from $\boldsymbol{S}$.
3. $\boldsymbol{DO}$ computes $r \leftarrow \mathsf{Dec}^{\mathsf{IPFE}}(c_{\boldsymbol{x}}, sk_{\boldsymbol{g}}^{\mathsf{IPFE}}) - \langle \boldsymbol{u}, \boldsymbol{g} \rangle$.

$\mathbf{PQuery}_{\boldsymbol{A},\boldsymbol{S}}(sk_{\boldsymbol{y}^i}^{\mathsf{RIPFE}}; c_{\boldsymbol{x}})$:
1. $\boldsymbol{S}$ receives $sk_{y^i}^{\mathsf{RIPFE}}$ from $\boldsymbol{A}$.
2. $\boldsymbol{S}$ computes $r \leftarrow \mathsf{Dec}^{\mathsf{RIPFE}}(c_{\boldsymbol{x}}, sk_{y^i}^{\mathsf{RIPFE}})$.
3. $\boldsymbol{A}$ receives r.

Fig. 3. Private functional encryption scheme supporting inner product queries PIPFE

5.2 Correctness, Security and Privacy

We need to prove the full security of our PIPFE scheme.

Theorem 3. *Let* IPFE *be a correct inner product functional encryption scheme and let* RIPFE *be as described in Figure 2. Then the* PIPFE *described in Figure 3 is a correct private functional encryption scheme for static databases supporting the inner product family of queries with distribution D.*

Proof. For EQuery, the correctness of the base IPFE gives us that $\Pr[s \leftarrow \mathsf{Dec}(c_{\boldsymbol{x}}, sk_{\boldsymbol{g}}) \neq \langle \boldsymbol{x}+\boldsymbol{u}, \boldsymbol{g} \rangle] = \mathrm{negl}(\kappa)$ as long as these are distributed as follows $(\mathsf{param}, \mathsf{msk}) \leftarrow \mathsf{SetUp}(1^\kappa)$, $c_x \leftarrow \mathsf{Enc}(\mathsf{msk}, \boldsymbol{x})$ and $sk_{\boldsymbol{g}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \boldsymbol{g})$, conditions satisfied by the protocols SetUp and EQuery from PIPFE, which gives us the expected equality between the output of EQuery and $\langle \boldsymbol{x}, \boldsymbol{g} \rangle$.

For PQuery, we need to prove that the output of $\mathsf{PQuery}(sk_{\boldsymbol{y}^i}^{\mathsf{RIPFE}}, c_{\boldsymbol{x}})$ is computationally indistinguishable from $\langle \boldsymbol{x}, \boldsymbol{y} \rangle + e^i$ with $e^i \leftarrow D$ for all $i \in [Q]$, when c_x and $sk_{\boldsymbol{y}^i}^{\mathsf{RIPFE}}$ are generated in SetUp. From the SetUp defined in PIPFE, this comes from Proposition 1.

Theorem 4. *Let* IPFE *be a 1-*$\mathsf{SEL\text{-}SIM}$*-secure inner product functional encryption scheme and let* RIPFE *be as described in Figure 2. Then our construction* PIPFE *described in Figure 3 is a 1-database secure and private functional encryption scheme for static databases supporting the inner product family of queries with distribution* D*.*

Proof. First of all, for the simulation part of Definition 11 (Table 4), let us define the PPT simulator $\mathsf{Sim} = (\mathsf{SetUpSim}, \mathsf{EQuerySim})$ as follows where $\boldsymbol{Y}$ denotes the set of Q queries $\boldsymbol{y}^1, \ldots, \boldsymbol{y}^Q \in \mathcal{F}$ and $\mathsf{EncSim}, \mathsf{KeyGenSim}$ are the simulators for the RIPFE scheme. For the exact formulation of these simulators we refer to the proof of Theorem 2.

$\mathbf{SetUpSim}_{\mathcal{C},\mathcal{A}}(1^\kappa, \hat{\mathcal{F}}_\epsilon^{\ell,\boldsymbol{X},\boldsymbol{Y}}; \boldsymbol{Y})$:
1. $\mathcal{C}$ computes the following
   a. $(c_{\boldsymbol{x}}^*, \mathsf{st}') \leftarrow \mathsf{EncSim}(1^\kappa)$
   b. $sk_{\boldsymbol{y}^i}^* \leftarrow \mathsf{KeyGenSim}(\mathsf{st}', \hat{\boldsymbol{y}}^i)$ for $i \in [Q]$
2. $\mathcal{C}$ receives st'
3. $\mathcal{A}$ receives $c_{\boldsymbol{x}}^*$ and $sk_{\boldsymbol{y}^1}^* \ldots sk_{\boldsymbol{y}^Q}^*$.

$\mathbf{EQuerySim}_{\mathcal{C},\mathcal{A}}((\mathsf{st}', g); c_{\boldsymbol{x}}^*)$:
1. $\mathcal{C}$ retrieves $c_{\boldsymbol{x}}^*$ from $\mathcal{A}$

The indistinguishability between SetUp and $\mathsf{SetUpSim}$ comes directly from the indistinguishability of the EncSim and $\mathsf{KeyGenSim}$ simulators, which is proven in Theorem 2, while EQuery and $\mathsf{EQuerySim}$ are indistinguishable to the adversary since its view of the protocol does not change.

Finally, for the privacy mechanism part of Definition 11, from the definition of PIPFE, the mechanism in Equation 1 is the same as the mechanism in Equation 2. Consequently, the privacy comes directly from Theorem 1.

6 Implementation Considerations

6.1 Differential Privacy Considerations

The first choice is what distribution will be used for the privacy mechanism. Since the distribution D_ϵ must be over $\mathbb{Z}$, we take for the DP mechanism $\mathcal{M}'$ the geometric distribution, as described in [26]. More specifically, the mechanism $\mathcal{M}'$ with error distribution sampled from $D \sim \mathsf{Geo}(\exp(-\epsilon/\Delta))$ is a $(\epsilon, 0)$-DP mechanism, where Δ is the ℓ_1-sensitivity of the family of functions $\mathcal{F}$ and $\mathsf{Geo}(\cdot)$ refers to the two-sided geometric distribution. We refer to Appendix D for the proof. For the sampling of this two-sided geometric we use the fact that a two-sided geometric distribution is the substraction of two one-sided geometric distributions

as shown in Proposition 3.1 in [30], while the one-sided geometric is sampled through the same algorithm as the NumPy library [2] rewritten in C. Having the precise distribution allows us also to make estimates about utility, which in broad terms measures how close the noisy response is to the actual value.

It follows that the utility of our mechanism is $O\left(\frac{1}{\epsilon}\right) \cdot \Delta \cdot \log\left(\frac{2}{\delta}\right)$ for any fixed δ (for the proof we refer to Appendix D). This means that there is a linear relation between the sensitivity of our family of queries and the size of the noise. As such it would be ideal to control this sensitivity. In the case of the inner product functionality, the sensitivity of a family of vectors is the sum of the maximum coefficient of each of them, which is bounded by $Q \cdot Y$. So in general the bigger the coefficient the bigger the noise, which makes sense since the purpose of this noise is to blur the statistic, and bigger coefficient generally means bigger difference between neighbouring databases. In comparison to the work by Bakas et al. [11], given that they add noise to each coefficient, their utility depends on the size of the database ℓ instead[4].

6.2 Implementation Specifics

For a concrete implementation we have opted for the IPFE scheme over the ring $\mathbb{Z}$ from [9] (the scheme is shown in Appendix E), which is proven to be simulation sound under the DDH assumption [8]. The DDH-based constructions are actually the most efficient currently known. Note that the decryption algorithm will only be able to recover the value when computing the discrete logarithm is efficient, therefore the responds needs to be smaller than some bound $B = \mathsf{poly}(\kappa)$. Then, by using the baby-giant steps algorithm presented in [38] we can recover the discrete logarithm in $\tilde{O}(B^{1/2})$. For the practical implementation we will consider B to be $\approx 2^{40}$, so that the discrete logarithm is computed in ≈ 0.5s.

Using our notation, the query result will be at most $\ell \cdot X \cdot Y + \alpha$ with probability δ where α represents the noise size and is computed as in Proposition 3 in Appendix D, so we can take $\alpha = (Q \cdot Y)/\epsilon \cdot \log(2/\delta)$. We will assume the number of queries asked $Q = 16$, the bound for these queries $Y = 2^7$ and the parameters $\epsilon = 0.1$ and $\delta = 2^{-100}$ all constant, and variate the amount of database entries ℓ with the bound these entries X, while putting a lower bound of 16 bits to X. Note that these values can be changed while keeping $\ell \cdot X \cdot Y + \alpha \approx 2^{40}$ or increased if we can assume a longer computation time for the discrete logarithm and therefore decryption time. In the case of very large number of entries, the discrete logarithm computation time is no longer the dominating factor so the bound B could be increased without notable effects on the decryption computation time, this is the reason why the lower bound to $|X|$ makes sense.

The proof of concept implementation was written in C and using the library CiFEr [1]. Some optimization was done, given the fact that we use a secret-key scheme instead of a public-key one, which allows us to reduce computations during setup given that the secret key can be used in the encryption and the

[4] Note that the sensitivity of the summation query is 1.

Table 6. Efficiency values for RIPFE instantiation based in [8].

	ℓ	$\lvert X\rvert$	msk	$c_{\boldsymbol{x}}$	$sk_{\boldsymbol{y}}$
Sizes	10	24	3 MB	3 KB	1 KB
	100	21	3 MB	37 KB	1 KB
	1 000	18	4 MB	375 KB	1 KB
	10 000	16	13 MB	3 MB	1 KB
	100 000	16	112 MB	36 MB	1 KB
	1 000 000	16	1 GB	366 MB	1 KB

	ℓ	SetUp	Encrypt	KeyGen	Decrypt
Comp. time	10	2.4442 s	0.0199 s	0.0001 s	0.6717 s
	100	2.4832 s	0.1810 s	0.0001 s	0.6899 s
	1 000	2.4468 s	1.7230 s	0.0005 s	0.7506 s
	10 000	2.4533 s	17.2846 s	0.0060 s	1.2976 s
	100 000	2.5095 s	172.3905 s	0.0603 s	5.0688 s
	1 000 000	3.1905 s	1720.3186 s	0.5974 s	24.8064 s

adversary has no access to this encryption key, therefore, precomputations for fast exponentiations can be performed during set up. More concretely, we apply the fixed-base comb method for fast exponentiations [34, Chapter 14, Section 14.6.3 iii] in the exponentiations used during the encryption (with precomputations done during set up) and the wNAF-based interleaving exponentiation method for fast simultaneous multiple exponentiation [36, Section 3.2] for the multiple exponentiations during the decryption algorithm.

We give in Table 6 the resulting values when run for different values of ℓ for 128-bit security (assuming a 3072 bit RSA modulus, as recommended by the NIST). The code was executed with Intel® Core™ i7-9800X (3.8GHz). Note that due to the form of the construction (see Appendix E) the encryption algorithm is easily parallelisable, so the timings could be easily reduced by using more cores. The timings grow linearly with the number of entries of the database ℓ for all algorithms, which is a lower bound for inner-product functional encryption schemes, so the overcost to obtain computational DP is quite low. In the case of the decryption and the set up algorithms the linear growth is overshadowed for small ℓ by computations that are constant by choice of the parameters. In case of the decryption algorithm this is the discrete logarithm computation for $\ell < 100\ 000$ while in case of the set up it is the precomputations for the fixed-base comb algorithm for $\ell < 1\ 000\ 000$. With these values we show the practical utility of our construction.

7 Conclusion

Our results may inspire follow-up works on the subject. Reducing the requirements for the reductions from simulation-based security to indistinguishability-based security would be an intriguing challenge. Moreover, it would tackle the

issue of multiple ciphertexts, as in the indistinguishability setting, security for one ciphertext usually implies security for multiple ciphertexts.

Another direction is to extend these results to the dynamic database setting, which would vastly open the implementation possibilities. However, due to the nature of dynamic databases, where the analyst can reuse the functional key to retrieve the noisy statistic from the updated database, adaptive simulation security for an unbounded number of ciphertexts might be required. In the case of inner product functional encryption, this is not possible, as mentioned in [8]. Furthermore, most efficient privacy mechanisms for dynamic databases rely on adding noise for each change to the database. As a result, some form of homomorphism must be implemented in conjunction with our design.

Acknowledgements We thank Bastien Vialla, Nicolas Desmoulins and Maxime Bélair for their help with the implementation of the scheme and its evaluation.

References

1. Cifer - functional encryption library, https://github.com/fentec-project/CiFEr
2. Numpy, https://github.com/numpy/numpy/tree/main
3. Abdalla, M., Bourse, F., Caro, A.D., Pointcheval, D.: Simple functional encryption schemes for inner products. In: Katz, J. (ed.) Public-Key Cryptography – PKC 2015. pp. 733–751. Springer, Berlin, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46447-2_33
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.) Advances in Cryptology – CRYPTO 2018. pp. 597–627. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-96884-1_20
5. Abdalla, M., Gay, R., Raykova, M., Wee, H.: Multi-input inner-product functional encryption from pairings. In: Coron, J.S., Nielsen, J.B. (eds.) Advances in Cryptology – EUROCRYPT 2017. pp. 601–626. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56620-7_21
6. Agarwal, A., Herlihy, M., Kamara, S., Moataz, T.: Encrypted databases for differential privacy. Proceedings on Privacy Enhancing Technologies **2019**(3), 170–190 (2019). https://doi.org/10.2478/popets-2019-0042
7. Agrawal, S., Wu, D.J.: Functional encryption: deterministic to randomized functions from simple assumptions. In: Coron, J.S., Nielsen, J.B. (eds.) Advances in Cryptology – EUROCRYPT 2017. pp. 30–61. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56614-6_2
8. Agrawal, S., Libert, B., Maitra, M., Titiu, R.: Adaptive simulation security for inner product functional encryption. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) Public-Key Cryptography – PKC 2020. pp. 34–64. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45374-9_2
9. Agrawal, S., Libert, B., Stehlé, D.: Fully secure functional encryption for inner products, from standard assumptions. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology – CRYPTO 2016. pp. 333–362. Springer, Berlin, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53015-3_12

10. Alwen, J., Barbosa, M., Farshim, P., Gennaro, R., Gordon, S.D., Tessaro, S., Wilson, D.A.: On the relationship between functional encryption, obfuscation, and fully homomorphic encryption. In: Stam, M. (ed.) Cryptography and Coding. pp. 65–84. Springer, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-45239-0_5
11. Bakas, A., Michalas, A., Dimitriou, T.: Private lives matter: A differential private functional encryption scheme. In: Proceedings of the Twelfth ACM Conference on Data and Application Security and Privacy. p. 300–311. Association for Computing Machinery, New York, NY (2022). https://doi.org/10.1145/3508398.3511514
12. Beimel, A., Nissim, K., Omri, E.: Distributed private data analysis: Simultaneously solving how and what. In: Wagner, D. (ed.) Advances in Cryptology – CRYPTO 2008. pp. 451–468. Springer, Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85174-5_25
13. Bishop, A., Jain, A., Kowalczyk, L.: Function-hiding inner product encryption. In: Iwata, T., Cheon, J.H. (eds.) Advances in Cryptology – ASIACRYPT 2015. pp. 470–491. Springer, Berlin, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48797-6_20
14. Boneh, D., Sahai, A., Waters, B.: Functional encryption: Definitions and challenges. In: Ishai, Y. (ed.) Theory of Cryptography. pp. 253–273. Springer, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19571-6_16
15. Bun, M., Chen, Y.H., Vadhan, S.: Separating computational and statistical differential privacy in the client-server model. In: Hirt, M., Smith, A. (eds.) Theory of Cryptography. pp. 607–634. Springer, Berlin, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53641-4_23
16. Chan, T.H.H., Shi, E., Song, D.: Private and continual release of statistics. ACM Trans. Inf. Syst. Secur. **14**(3) (nov 2011). https://doi.org/10.1145/2043621.2043626
17. Desfontaines, D., Pejó, B.: Sok: differential privacies. Proceedings on privacy enhancing technologies **2020**(2), 288–313 (2020). https://doi.org/10.2478/popets-2020-0028
18. Ding, B., Kulkarni, J., Yekhanin, S.: Collecting telemetry data privately. Advances in Neural Information Processing Systems **30** (2017), https://proceedings.neurips.cc/paper_files/paper/2017/hash/253614bbac999b38b5b60cae531c4969-Abstract.h
19. Dwork, C.: Differential privacy. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) Automata, Languages and Programming. pp. 1–12. Springer, Berlin, Heidelberg (2006). https://doi.org/10.1007/11787006_1
20. Dwork, C., McSherry, F., Nissim, K., Smith, A.: Calibrating noise to sensitivity in private data analysis. In: Halevi, S., Rabin, T. (eds.) Theory of Cryptography. pp. 265–284. Springer, Berlin, Heidelberg (2006). https://doi.org/10.1007/11681878_14
21. Dwork, C., Roth, A.: The algorithmic foundations of differential privacy. Foundations and Trends® in Theoretical Computer Science **9**(3–4), 211–407 (2014). https://doi.org/10.1561/0400000042, http://dx.doi.org/10.1561/0400000042
22. Erlingsson, U., Pihur, V., Korolova, A.: Rappor: Randomized aggregatable privacy-preserving ordinal response. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. p. 1054–1067. Association for Computing Machinery, New York, NY (2014). https://doi.org/10.1145/2660267.2660348

23. Garfinkel, S.L., Abowd, J.M., Powazek, S.: Issues encountered deploying differential privacy. In: Proceedings of the 2018 Workshop on Privacy in the Electronic Society. p. 133–137. Association for Computing Machinery, New York, NY (2018). https://doi.org/10.1145/3267323.3268949
24. Garg, S., Gentry, C., Halevi, S., Zhandry, M.: Functional encryption without obfuscation. Cryptology ePrint Archive, Paper 2014/666 (2014), https://eprint.iacr.org/2014/666
25. Ghazi, B., Ilango, R., Kamath, P., Kumar, R., Manurangsi, P.: Separating computational and statistical differential privacy (under plausible assumptions) (2022). https://doi.org/10.48550/arXiv.2301.00104
26. Ghosh, A., Roughgarden, T., Sundararajan, M.: Universally utility-maximizing privacy mechanisms. In: Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing. p. 351–360. Association for Computing Machinery, New York, NY, USA (2009). https://doi.org/10.1145/1536414.1536464
27. Goyal, V., Jain, A., Koppula, V., Sahai, A.: Functional encryption for randomized functionalities. In: Dodis, Y., Nielsen, J.B. (eds.) Theory of Cryptography. pp. 325–351. Springer, Berlin, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46497-7_13
28. Hamdi, A.: Functional encryption for blind external data processing. Ph.D. thesis, Université de Lyon (2021), https://theses.hal.science/tel-03500313/
29. Hsu, J., Gaboardi, M., Haeberlen, A., Khanna, S., Narayan, A., Pierce, B.C., Roth, A.: Differential privacy: An economic method for choosing epsilon. In: 2014 IEEE 27th Computer Security Foundations Symposium. pp. 398–410. IEEE, Vienna, Austria (2014). https://doi.org/10.1109/CSF.2014.35
30. Inusah, S., Kozubowski, T.J.: A discrete analogue of the laplace distribution. Journal of Statistical Planning and Inference **136**(3), 1090–1102 (2006). https://doi.org/10.1016/j.jspi.2004.08.014
31. Komargodski, I., Segev, G., Yogev, E.: Functional encryption for randomized functionalities in the private-key setting from minimal assumptions. Journal of Cryptology **31**(1), 60–100 (2018). https://doi.org/10.1007/s00145-016-9250-8
32. Li, C.: Optimizing Linear Queries Under Differential Privacy. Ph.D. thesis, University of Massachusetts Amherst (2013), https://doi.org/10.7275/9seb-w353
33. McGregor, A., Mironov, I., Pitassi, T., Reingold, O., Talwar, K., Vadhan, S.: The limits of two-party differential privacy. In: 2010 IEEE 51st Annual Symposium on Foundations of Computer Science. pp. 81–90. IEEE, Las Vegas, NV (2010). https://doi.org/10.1109/FOCS.2010.14
34. Menezes, A.J., Vanstone, S.A., Oorschot, P.C.V.: Handbook of Applied Cryptography. CRC Press, Inc., Boca Raton FL, USA (1996)
35. Mironov, I., Pandey, O., Reingold, O., Vadhan, S.: Computational differential privacy. In: Halevi, S. (ed.) Advances in Cryptology - CRYPTO 2009. pp. 126–142. Springer, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03356-8_8
36. Möller, B.: Algorithms for multi-exponentiation. In: Vaudenay, S., Youssef, A.M. (eds.) Selected Areas in Cryptography. pp. 165–180. Springer Berlin Heidelberg, Berlin, Heidelberg (2001). https://doi.org/10.1007/3-540-45537-X_13
37. O'Neill, A.: Definitional issues in functional encryption. Cryptology ePrint Archive, Paper 2010/556 (2010), https://eprint.iacr.org/2010/556
38. Pollard, J.M.: Kangaroos, monopoly and discrete logarithms. J. Cryptol. **13**(4), 437–447 (jan 2000). https://doi.org/10.1007/s001450010010
39. Schwartz, M.D., Denning, D.E., Denning, P.J.: Linear queries in statistical databases. ACM Trans. Database Syst. **4**(2), 156–167 (jun 1979). https://doi.org/10.1145/320071.320073

40. Shoaran, M., Thomo, A., Weber, J.: Differential privacy in practice. In: Jonker, W., Petković, M. (eds.) Secure Data Management. pp. 14–24. Springer Berlin Heidelberg, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32873-2_2

A Definitions for Functional Encryption

Generic functional encryption schemes can be defined as private key or public key. We will use the private key version to better conform with the model presented in Figure 1. Since we are considering a sole database owner protecting their database outsourced to the server, it makes little sense to allow for other entities to encrypt over the database. For the purpose of this section we will consider $\mathcal{X}$ to be a database space, $\mathcal{S}$ an output space and $\mathcal{F}$ a family of deterministic functions $f : \mathcal{X} \to \mathcal{S}$.

Definition 12 (Adapted from Definition 2.3, [3]). *Let $\kappa \in \mathbb{N}_{>0}$ be a security parameter. We define a* secret-key functional encryption scheme *supporting the family of functions $\mathcal{F}$ the following tuple of* PPT *algorithms:*

- $\mathsf{SetUp}(1^\kappa, \mathcal{F})$: *given the security parameter as input, it outputs some public parameters* param *and a master secret key* msk*. We will assume the public parameters as inputs in all other algorithms.*
- $\mathsf{Enc}(\mathsf{msk}, x)$: *given the master secret key* msk *and a plaintext $x \in \mathcal{X}$ as inputs, it outputs a ciphertext c_x.*
- $\mathsf{KeyGen}(\mathsf{msk}, f)$: *given the master secret key* msk *and a function $f \in \mathcal{F}$ as inputs, it outputs a functional key sk_f.*
- $\mathsf{Dec}(c_x, sk_f)$: *given a ciphertext c_x and a functional key sk_f as inputs, it outputs a string s.*

As usual with encryption schemes, there is a correctness notion, which follows the standard definitions of correctness for encryption: for any plaintext x and function $f \in \mathcal{F}$, then $\Pr[s \leftarrow \mathsf{Enc}(c_x, sk_f) \neq f(x)] = \mathrm{negl}(\kappa)$ where the probability is taken over $(\mathsf{param}, \mathsf{msk}) \leftarrow \mathsf{SetUp}(1^\kappa, \mathcal{F})$, $c_x \leftarrow \mathsf{Dec}(\mathsf{msk}, x)$ and $sk_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$.

Also, as with correctness, there are two main security definitions analogous to those for encryption: indistinguishability and simulation security. However, unlike in public key encryption, these two definitions are not equivalent, specifically, indistinguishability-based security does not imply simulation-based security. This was already discussed by Boneh, Sahai and Waters in [14] and O'Neill in [37] which leads to a plethora of different security definitions.

Another notable difference in the definitions is due to the appearance of the functional keys which are independent to the ciphertexts. As such there is a distinction in when the adversary is allowed to ask for functional keys. We say an adversary $\mathcal{A}$ is *selective* if it is only allowed to ask for functional keys after setting the challenge, while we say $\mathcal{A}$ is *adaptive* if, on top of that, it can also ask for functional keys before setting the challenge. There has been plenty of study about possibility and impossibility of each type of security (selective or adaptive,

indistinguishability or simulation based) for generic functional encryption with positive and negative results. Therefore, it must be evaluated in a case by case basis for each type of functionality. In the case of inner product functional encryption it is well-known (as mentioned in [8]) that adaptive simulation security for an unbounded number of ciphertexts is not possible.

In this work we focus on selective simulation security against one challenge ciphertext for secret key functional encryption, so let us give its definition.

Definition 13 (Adapted from Section 2.3, [8]). *Let $\kappa \in \mathbb{N}_{>0}$ be a security parameter and let* $\mathsf{FE} = (\mathsf{SetUp}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ *be a functional encryption scheme for the function family $\mathcal{F}$. We say* FE *is* 1-SEL-SIM secure *if there exists a* PPT *simulator* $\mathsf{Sim} = (\mathsf{EncSim}, \mathsf{KeyGenSim})$ *such that for every* PPT *adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, the outputs of the real and ideal experiments experiments (see Table 7) are computationally indistinguishable, where the oracles are described as follows.*

Table 7. Real and ideal experiments in 1-SEL-SIM security for FE.

$\mathsf{Exp}^{real}_{\mathcal{A}}(1^\kappa, \mathcal{F})$

1: $(x, \mathsf{st}_1) \leftarrow \mathcal{A}_1$ where $x \in \mathcal{X}$
2: $(\mathsf{param}, \mathsf{msk}) \leftarrow \mathsf{SetUp}(1^\kappa, \mathcal{F})$
3: $c_x \leftarrow \mathsf{Enc}(x, \mathsf{msk})$
4: $\gamma \leftarrow \mathcal{A}_2^{\mathcal{O}_1(\mathsf{msk},\cdot),\mathcal{O}_2(\cdot,\cdot)}(c_x, \mathsf{st}_1)$
Output: $(x, \{f\}, \{sk_f\}, \{s\}, \gamma)$

$\mathsf{Exp}^{ideal}_{\mathcal{A},\mathsf{Sim}}(1^\kappa, \mathcal{F})$

1: $(x, \mathsf{st}_1) \leftarrow \mathcal{A}_1$ where $x \in \mathcal{X}$
2: $(\mathsf{param}, c^*_x, \mathsf{st}') \leftarrow \mathsf{EncSim}(1^\kappa, \mathcal{F})$
3: $\gamma \leftarrow \mathcal{A}_2^{\mathcal{O}'_1(\mathsf{st}',\cdot),\mathcal{O}'_2(\cdot,\cdot)}(c^*_x, \mathsf{st}_1)$
Output: $(x, \{f'\}, \{sk'_f\}, \{s'\}, \gamma)$

1. ***Real Experiment:*** *$\mathcal{O}_1(\mathsf{msk}, \cdot)$ refers to the non-simulated key generation oracle $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$. The set $\{f\}$ denotes the key queries made by $\mathcal{A}_2$. $\mathcal{O}_2(\cdot, \cdot)$ refers to the decryption oracle $\mathsf{Dec}(\cdot, \cdot)$. The sets $\{sk_f\}, \{s\}$ denote the functional keys queried by $\mathcal{A}$ and the responses of the oracle respectively.*
2. ***Ideal Experiment:*** *$\mathcal{O}'_1(\mathsf{st}', \cdot)$ refers to the simulated key generation oracle $\mathsf{KeyGenSim}(\mathsf{st}', \cdot)$. The set $\{f'\}$ denotes the key queries made by $\mathcal{A}_2$. $\mathcal{O}'_2(\cdot, \cdot)$ refers to the decryption oracle $\mathsf{Dec}(\cdot, \cdot)$. The sets $\{sk'_f\}, \{s'\}$ denote the functional keys queried by $\mathcal{A}$ and the responses of the oracle respectively.*

B Discussion about other Privacy Enhancing Technologies

Privacy enhancing technologies (PETs) are a compendium of technologies including both functional encryption as well as differential privacy, so it begs the question whether other of these technologies could be used to fulfill the model at discussion, more specifically fully homomorphic encryption (FHE) and multi-party computation (MPC).

FHE encompasses encryption schemes with the property that one can operate on the ciphertexts with a ring structure while maintaining the structure of the

underlying plaintexts (i.e. the addition or multiplication of ciphertexts is a ciphertext of the corresponding operation over the plaintexts). As most encryption schemes, let them be public key or secret key, only the (private) secret key can be used for decrypting *any* ciphertext (either pre or post computation) and this leads to an issue in our model from Figure 1, since the decryption will need to be done by the database owner. The scheme would be as follows for a database $\boldsymbol{x} = (x^1, \ldots, x^\ell)$ and linear query $\boldsymbol{y} = (y^1, \ldots, y^\ell)$, the database owner sends the encrypted database to the server (note that in this case the ciphertext will need to be cut into pieces $c_{x^1}, \ldots, c_{x^\ell}$). Then whenever an analyst has a query they send it to the database owner who encrypts to get $c_{y^1}, \ldots, c_{y^\ell}$. The database owner sends the ciphertexts to the server who computes $c_{\langle \boldsymbol{x}, \boldsymbol{y} \rangle} = \sum c_{x^i} \cdot c_{y^i}$ and returns it to the database owner. The database owner decrypts the ciphertext and samples some noise before sending the noisy statistic to the analyst, as it is shown in Figure 4.

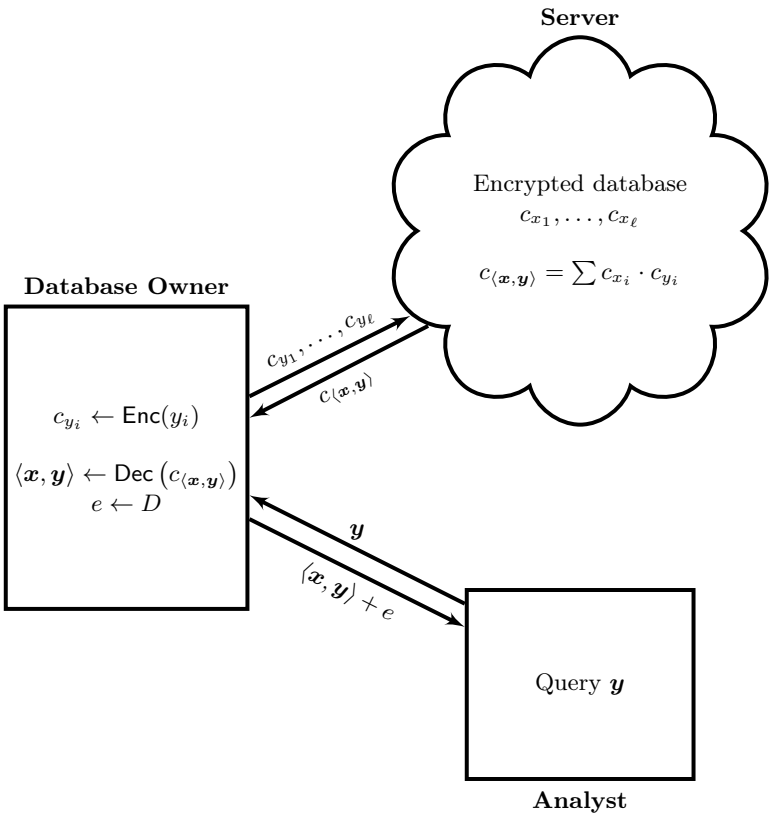

Fig. 4. Diagram of interactions for the FHE based solution.

Note that in this case there is no interaction between the server and the analyst, thus defeating the purpose of FHE, since the database owner could send an encrypted database through a regular encryption scheme and for each query retrieve and decrypt the ciphertext, to then compute the query and the noise for answering the analyst.

MPC includes very general forms of computation, usually revolving around secret sharing and the computation of those secrets under specific circumstances. The drawback of using MPC is usually the high level of interaction between the several parties to allow for the recovery of the secret. In this case, a 2PC protocol could be conceived where one same secret share could be used to recover different functions of the database when interacting with different shares from the other party. In such position then, referencing Figure 1, step 1 would be sending this "privileged" share related to the database, while step 2 would entail the computation of a share corresponding to the query. Finally step 3 would be the 2PC protocol mentioned before.

C A Trivial Scheme

In this appendix we give a trivial scheme that satisfies simulation security by the definition in [27] but does not satisfy our definition. This is due to the fact that the randomness is given out with the functional key. Let $\mathcal{F}$ and $\hat{\mathcal{F}}$ be defined as in Section 3 for some distribution D and $\mathsf{FE} = (\mathsf{SetUp}^{\mathsf{FE}}, \mathsf{Enc}^{\mathsf{FE}}, \mathsf{KeyGen}^{\mathsf{FE}}, \mathsf{Dec}^{\mathsf{FE}})$ be a functional encryption scheme for the family of functions $\mathcal{F}$. We define the following randomized functional encryption scheme $\mathsf{RFE} = (\mathsf{SetUp}^{\mathsf{RFE}}, \mathsf{Enc}^{\mathsf{RFE}}, \mathsf{KeyGen}^{\mathsf{RFE}}, \mathsf{Dec}^{\mathsf{RFE}})$ for the family of randomized functions $\hat{\mathcal{F}}$. See Figure 5.

$\mathbf{SetUp}^{\mathbf{RFE}}(\mathbf{1}^{\kappa})$:
$(\mathsf{msk}^{\mathsf{FE}}, param^{\mathsf{FE}}) \leftarrow \mathsf{SetUp}^{\mathsf{FE}}(1^{\kappa})$
$Output\ (\mathsf{msk}^{\mathsf{RFE}}, param^{\mathsf{RFE}}) = (\mathsf{msk}^{\mathsf{FE}}, param^{\mathsf{FE}})$

$\mathbf{Enc}^{\mathbf{RFE}}(\mathbf{msk}^{\mathbf{RFE}}, \boldsymbol{x})$:
$c_x \leftarrow \mathsf{Enc}^{\mathsf{FE}}(\mathsf{msk}^{\mathsf{FE}}, x)$
$Output\ c_x$

$\mathbf{KeyGen}^{\mathbf{RFE}}(\mathbf{msk}^{\mathbf{RFE}}, \hat{\boldsymbol{f}})$:
$e_f \leftarrow D$
$sk_f \leftarrow \mathsf{KeyGen}^{\mathsf{FE}}(\mathsf{msk}^{\mathsf{FE}}, f)$
$Output\ sk_{\hat{f}}^{\mathsf{RFE}} = (e_f, sk_{\hat{f}})$

$\mathbf{Dec}^{\mathbf{RFE}}(\boldsymbol{c_x}, \boldsymbol{sk}_{\hat{\boldsymbol{f}}}^{\mathbf{RFE}})$:
$f(x) \leftarrow \mathsf{Dec}^{\mathsf{FE}}(c_x, sk_f)$
$s \leftarrow f(x) + e_f$
$Output\ s$

Fig. 5. Trivial RFE scheme

It is straight-forward to see that as long as FE is 1-SEL-SIM secure, RFE will also be 1-SEL-SIM under the definition in [27]. The simulators are obtained by substituting the FE algorithms by their simulators and, since the master secret key is only used in those algorithms, the simulators will hold for RFE. Let $\mathsf{Sim}^{\mathsf{FE}} = (\mathsf{EncSim}^{\mathsf{FE}}, \mathsf{KeyGenSim}^{\mathsf{FE}})$ be the simulators for the FE scheme, we construct the following RFE simulators $\mathsf{Sim}^{\mathsf{RFE}} = (\mathsf{EncSim}^{\mathsf{RFE}}, \mathsf{KeyGenSim}^{\mathsf{RFE}})$.

$\mathbf{EncSim}^{\mathbf{RFE}}(\mathbf{1}^{\kappa})$:
$(\mathsf{param}, c_x^*, \mathsf{st}') \leftarrow \mathsf{EncSim}^{\mathsf{FE}}(1^{\kappa})$
Output $(\mathsf{param}, c_x^*, \mathsf{st}')$

$\mathbf{KeyGen}^{\mathbf{RFE}}(\mathbf{st}', \hat{\boldsymbol{f}})$:
$e_f \leftarrow D$
$sk_f^* \leftarrow \mathsf{KeyGenSim}^{\mathsf{FE}}(\mathsf{st}', f)$
Output $sk_{\hat{f}}^{*\mathsf{RFE}} = \left(e_f, sk_{\hat{f}}^*\right)$

This counter-example is for 1-SEL-SIM security so that the randomized function $\hat{f}$ complies with the definition of randomized function. It can be extended to N-SEL-SIM security by adding noise to the encryption too.

However, this scheme is not 1-SEL-SIM under our definition since the key generation simulator is unable to "extract" e_f from a value v^f received from KeyIdeal given the information it has access to. The only way to obtain e_f from v^f would be to know the challenge plaintexts, but from Definition 8 it is clear that the key generation simulator has no access to the challenges. Therefore, any scheme which gives out the randomness in the functional key will not satisfy simulation security under our definition, which is desirable for our use-case. This means that no randomized functional encryption scheme giving out the noise in the functional key will be 1-SEL-SIM under our definition as we wanted.

D Privacy and utility of the Geometric mechanism

Proposition 2. *Let $\mathcal{X}$ be a database space, $\mathcal{S} = \mathbb{Z}^Q$, $\mathcal{F}$ be a family of queries and let $\boldsymbol{f} \in \mathcal{F}^Q$. Let D be a random variable, $D \sim \mathsf{Geo}(\exp(-\epsilon/\Delta_{\boldsymbol{f}}))$. Then the geometric mechanism defined as*

$$\mathcal{M}_{\boldsymbol{f}}(x; r) := f_i(x) + e(r_i), \text{ for } i \in [Q]$$

where $x \in \mathcal{X}$ and $e(r_i) \leftarrow D$ is $(\epsilon, 0)$-DP.

Proof. Let $\boldsymbol{t} \in \mathbb{Z}^Q$, then

$$\begin{aligned}
\frac{\Pr[\mathcal{M}_{\boldsymbol{f}}(x) = \boldsymbol{t}]}{\Pr[\mathcal{M}_{\boldsymbol{f}}(x') = \boldsymbol{t}]} &= \prod_{i=1}^{Q} \frac{\Pr[f_i(x) + e(r_i) = t_i]}{\Pr[f_i(x') + e(r_i) = t_i]} \\
&= \prod_{i=1}^{Q} \frac{\exp\left(\frac{-\epsilon \cdot |t_i - f_i(x)|}{\Delta_{\boldsymbol{f}}}\right)}{\exp\left(\frac{-\epsilon \cdot |t_i - f_i(x')|}{\Delta_{\boldsymbol{f}}}\right)} \\
&\leq \prod_{i=1}^{Q} \exp\left(\frac{\epsilon}{\Delta_{\boldsymbol{f}}} |f_i(x') - f_i(x)|\right) \\
&= \exp\left(\frac{-\epsilon}{\Delta_{\boldsymbol{f}}} \|\boldsymbol{f}(x') - \boldsymbol{f}(x)\|_1\right) \\
&\leq \exp(\epsilon).
\end{aligned}$$

Now adding for all $\boldsymbol{t} \in S$

$$\begin{aligned}
\frac{\Pr[\mathcal{M}_{\boldsymbol{f}}(x) \in S]}{\Pr[\mathcal{M}_{\boldsymbol{f}}(x') \in S]} &= \frac{\sum_{\boldsymbol{t} \in S} \Pr[\mathcal{M}_{\boldsymbol{f}}(x) = \boldsymbol{t}]}{\sum_{\boldsymbol{t} \in S} \Pr[\mathcal{M}_{\boldsymbol{f}}(x') = \boldsymbol{t}]} \\
&\leq \frac{\sum_{\boldsymbol{t} \in S} e^{\epsilon} \cdot \Pr[\mathcal{M}_{\boldsymbol{f}}(x') = \boldsymbol{t}]}{\sum_{\boldsymbol{t} \in S} \Pr[\mathcal{M}_{\boldsymbol{f}}(x') = \boldsymbol{t}]} \\
&= e^{\epsilon}
\end{aligned}$$

as we wanted.

Note that $\Delta_{\boldsymbol{f}}$ can be changed by any bigger constant, so by using Δ we eliminate the dependance on $\boldsymbol{f}$.

Proposition 3. *The Geometric mechanism as described in Proposition 2 is* $\left(O\left(\frac{1}{\epsilon}\right) \cdot \Delta \cdot \log\left(\frac{2}{\delta}\right), \delta\right)$*-useful.*

Proof. We want to find for any given δ, the corresponding minimum α in regards to utility. As such we need to solve the inequation for an $1-\delta$, which is the same as inverting the inequality inside the probability and compare to δ inverting the equality outside too. Note that substracting $f(x)$ to the mechanism will always leave us just a sample of the geometric distribution $D \sim \mathsf{Geo}(\epsilon/\Delta)$.

$$\begin{aligned}
\Pr\left[|D| \geq \alpha\right] &= 2 \cdot \sum_{k=\alpha}^{\infty} \Pr\left[D = k\right] \\
&= 2 \cdot \frac{1 - \exp\left(\frac{-\epsilon}{\Delta}\right)}{1 + \exp\left(-\frac{\epsilon}{\Delta}\right)} \sum_{k=\alpha}^{\infty} \exp\left(\frac{-\epsilon}{\Delta}\right)^k \\
&= 2 \cdot \frac{1 - \exp\left(\frac{-\epsilon}{\Delta}\right)}{1 + \exp\left(\frac{-\epsilon}{\Delta}\right)} \cdot \frac{\exp\left(\frac{-\epsilon}{\Delta}\right)^{\alpha}}{1 - \exp\left(\frac{-\epsilon}{\Delta}\right)} \\
&\leq \delta
\end{aligned}$$

Which in turn gives us

$$\alpha \geq \frac{\Delta}{\epsilon} \cdot \left(\log\left(\frac{2}{\delta}\right) - \log\left(1 + \exp\left(\frac{-\epsilon}{\Delta}\right)\right)\right)$$
$$\approx O\left(\frac{1}{\epsilon}\right) \cdot \Delta \cdot \log\left(\frac{2}{\delta}\right)$$

as we wanted to see.

E Implemented Scheme

We present the DDH-based scheme first proposed in [9] and proven simulation sound against selective adversaries in [5] and against adaptive adversaries in [8]. This scheme is public-key, but we transform it into secret-key by incorporating the master public key into the master private key.

Encryption Scheme 1 (Adapted from Section 3, [8])

- **SetUp**$(1^\kappa, \mathcal{F}^{\ell,X,Y})$: *Choose a cyclic group* $\mathbb{G}$ *of prime order* $q > 2^\kappa$ *and two generators* $g, h \xleftarrow{\$} \mathbb{G}$. *Then for all* $i \in [\ell]$ *sample* $s_i, t_i \xleftarrow{\$} \mathbb{Z}_q$ *and compute* $h_i = g^{s_i} \cdot h^{t_i}$. *Define*

$$\mathsf{param} = (X, Y, \ell, \mathbb{G}, g, h) \text{ and } \mathsf{msk} = \{s_i, t_i, h_i\}_{i\in[\ell]}$$

- **Enc**$(\mathsf{msk}, \boldsymbol{x})$: *To encrypt a vector* $\boldsymbol{x} \in \mathbb{Z}_q^\ell$ *with* $\|\boldsymbol{x}\|_\infty < X$, *sample* $r \xleftarrow{\$} \mathbb{Z}_q$ *and compute*

$$C = g^r, \quad D = h^r, \quad \{E_i = g^{x_i} \cdot h_i^r\}_{i\in[\ell]}.$$

  Output $c_{\boldsymbol{x}} = (C, D, E_1, \ldots, E_\ell)$.
- **KeyGen**$(\mathsf{msk}, \boldsymbol{y})$: *To compute a functional decryption key for the vector* $\boldsymbol{y}$ *with* $\|\boldsymbol{y}\|_\infty < Y$ *compute* $s_{\boldsymbol{y}} = \langle \boldsymbol{s}, \boldsymbol{y}\rangle$ *and* $t_{\boldsymbol{y}} = \langle \boldsymbol{t}, \boldsymbol{y}\rangle$. *Output* $sk_{\boldsymbol{y}} = (s_{\boldsymbol{y}}, t_{\boldsymbol{y}})$.
- **Dec**$(c_{\boldsymbol{x}}, sk_{\boldsymbol{y}})$: *Given a ciphertext* $c_{\boldsymbol{x}} = (C, D, E_1, \ldots, E_\ell)$ *and a functional decryption key* $sk_{\boldsymbol{y}} = (s_{\boldsymbol{y}}, t_{\boldsymbol{y}})$ *compute*

$$E_{\boldsymbol{y}} = \frac{\prod_{i=1}^{\ell} E_i^{y_i}}{C^{s_{\boldsymbol{y}}} \cdot D^{t_{\boldsymbol{y}}}}.$$

  Finally compute $s = \log_q(E_{\boldsymbol{y}})$ *and output* s.

This scheme is proven correct and secure in [8] (Section 3, Theorem 1), and satisfies the two-step decryption property (Property 1). More in particular, the function $\mathcal{E}$ is computing the power of the generator $\mathcal{E}(\gamma, noise) = g^\gamma$, therefore the PPT algorithm Dec2 is the baby-giant steps algorithm to compute the discrete logarithm and the PPT algorithm $\mathsf{Dec1}^{\mathsf{IPFE}}$ is computing the value $E_{\boldsymbol{y}}$.

However, it is a public key scheme and for our construction only a secret key scheme is needed. Therefore we can slightly change it to reduce the amount of

computations needed by using the master secret key in the encryption algorithm. On top of that, we implemented the fixed-base comb method for fast exponentiation, and we consider the precomputations needed F_g, F_h part of the public parameters. As such, the randomized encryption scheme we have implemented is as follows.

Encryption Scheme 2

- $\mathbf{SetUp}(1^\kappa, \hat{\mathcal{F}}^{\ell,X,Y})$: *Define* D_ϵ *as* $\mathsf{Geo}(-\epsilon/\Delta)$, *and as such* $\alpha = (\kappa \cdot \Delta)/\epsilon$. *Choose a cyclic group* $\mathbb{G}$ *of prime order* $q > 2^\kappa$ *with operation* $\circ$ *and two generators* $g, h \xleftarrow{\$} \mathbb{G}$ *and compute* F_g, F_h *the precomputations for the fixed-base com method for fast exponentiation. Set* $L = q$ *and sample* $\boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^\ell$. *Then for all* $i \in [\ell]$ *sample* $s_i, t_i \xleftarrow{\$} \mathbb{Z}_q$. *Define*

$$\mathsf{param} = (X, Y, \ell, \mathbb{G}, g, h, F_g, F_h) \text{ and } \mathsf{msk} = (\boldsymbol{u}, \{s_i, t_i\}_{i\in[\ell]})$$

- $\mathbf{Enc}(\mathsf{msk}, \boldsymbol{x})$: *To encrypt a vector* $\boldsymbol{x} \in \mathbb{Z}_q^\ell$ *with* $\|\boldsymbol{x}\|_\infty < X$, *compute* $\boldsymbol{d} = \boldsymbol{x} + \boldsymbol{u} \pmod q$ *and then sample* $r \xleftarrow{\$} \mathbb{Z}_q$ *and compute,*

$$C = g^r, \quad D = h^r, \quad \{E_i = g^{d_i + s_i \cdot r} \cdot h^{t_i \cdot r}\}_{i\in[\ell]}.$$

  Output $c_{\boldsymbol{d}} = (C, D, E_1, \ldots, E_\ell)$.
- $\mathbf{KeyGen}(\mathsf{msk}, \boldsymbol{y})$: *To compute a functional decryption key for the vector* $\boldsymbol{y}$ *with* $\|\boldsymbol{y}\|_\infty < Y$, *first sample* $e_{\boldsymbol{y}} \leftarrow \mathsf{Geo}(-\epsilon/\Delta)$ *and* $u'_{\boldsymbol{y}} \xleftarrow{\$} \mathbb{Z}_q$. *Then compute* $d'_{\boldsymbol{y}} = e_{\boldsymbol{y}} + u'_{\boldsymbol{y}} \pmod q$ *and* $zk_{\boldsymbol{y}} = \langle \boldsymbol{u}, \boldsymbol{y} \rangle + u'_{\boldsymbol{y}} \pmod q$. *Finally, compute* $s_{\boldsymbol{y}} = \langle \boldsymbol{s}, \boldsymbol{y} \rangle$ *and* $t_{\boldsymbol{y}} = \langle \boldsymbol{t}, \boldsymbol{y} \rangle$ *and output* $sk_{\boldsymbol{y}} = (d'_{\boldsymbol{y}}, s_{\boldsymbol{y}}, t_{\boldsymbol{y}}, zk_{\boldsymbol{y}})$.
- $\mathbf{Dec}(c_{\boldsymbol{d}}, sk_{\boldsymbol{y}})$: *Given a ciphertext* $c_{\boldsymbol{d}} = (C, D, E_1, \ldots, E_\ell)$ *and a functional decryption key* $sk_{\boldsymbol{y}} = (s_{\boldsymbol{y}}, t_{\boldsymbol{y}})$ *compute*

$$E_{\boldsymbol{y}} = \frac{\prod_{i=1}^{\ell} E_i^{y_i}}{C^{s_{\boldsymbol{y}}} \cdot D^{t_{\boldsymbol{y}}}}.$$

  Finally compute $s = \log_q(E_{\boldsymbol{y}} \circ g^{d'_{\boldsymbol{y}} - zk_{\boldsymbol{y}}})$ *and output* s.