This extended abstract appeared in
Proceedings of the 2017 ACM Symposium on Information, computer and communications security (AsiaCCS '17)
(April 4–6, 2017, Abu Dhabi, United Arab Emirates), ACM Press. DOI: 10.1145/3052973.3052996

Functional Encryption with Oblivious Helper

Pierre-Alain Dupont and David Pointcheval

CNRS, INRIA, and ENS/PSL Research University, Paris, France

Abstract. Functional encryption is a nice tool that bridges the gap between usability and privacy when providing access to huge databases: while being encrypted, aggregated information is available with a fine-tuned control by the owner of the database who can specify the functions he allows users to compute on the data. Unfortunately, giving access to several functions might leak too much information on the database, since once the decryption capability is given for a specific function, this is for an unlimited number of ciphertexts. In the particular case of the inner-product, if rows or records of the database contain ℓ fields on which one got ℓ independent inner-product capabilities, one can extract all the individual fields. On the other hand, the major applications that make use of inner-products, such as machine-learning, need to compute many of them.
This paper deals with a practical trade-off in order to allow the computation of various inner-products, while still protecting the confidentiality of the data. To this aim, we introduce an *oblivious helper*, that will be required for any decryption-query, in order to control the leakage of information on the database. It should indeed learn just enough information to guarantee the *confidentiality* of the database, but without endangering the *privacy* of the queries.

Keywords: Functional encryption; oblivious decryption; confidentiality; privacy

1 Introduction

With the new *cloud* paradigm shift, and the rise of *machine learning*, very huge databases are being made available to many users, in order to allow computations on them. For example, this allows medical research to share quite useful data for improving medical diagnosis. But such endeavour should not sacrifice privacy. Individual fields in databases should not be released since they contain critical information on the people and only aggregated data should be provided as it is enough anyway for machine learning techniques.

Our goal is thus to allow aggregation of data from encrypted databases, in order to provide both privacy and utility. Unfortunately, classical encryption schemes do not address this issue, because they target specific users (the owners of the decryption key) who then have full ability to decrypt any data. On the other hand, the recent *Functional Encryption* (FE) [SW05,BSW11, O'N10] allows a better control of which can be decrypted.

In such a scheme, with the master secret key, the authority can provide, for any function $\boldsymbol{f}$, a functional key $\mathsf{sk}\boldsymbol{f}$ that allows, on a ciphertext C of a message $\boldsymbol{m}$, to get back $\boldsymbol{f}(\boldsymbol{m})$ only. No more information about $\boldsymbol{m}$ is released. We stress that functional encryption is used to encrypt large messages $\boldsymbol{m}$ (records with many fields of a database, or even the entire database) in each large ciphertext. Hence $\boldsymbol{m}$ in bold that denotes a vector. So $\boldsymbol{f}(\boldsymbol{m})$ can actually be some computation on records of the database, or on the entire database. And while the data are encrypted, the functional key $\mathsf{sk}\boldsymbol{f}$ allows to get the evaluation in clear but nothing else is leaked.

A simpler approach would be to ask for on-the-fly computation of $\boldsymbol{f}(\boldsymbol{m})$, but this requires that the data owners (which may be numerous in a public-key encryption scheme) are on-line and available to answer the requests. Another solution would be to directly encrypt $\boldsymbol{f}(\boldsymbol{m})$ under a key related to $\boldsymbol{f}$ (which is possible for any function encryption scheme), but this requires in-advance precise knowledge of the functions $\boldsymbol{f}$ that can be asked, as the ciphertext's size grows linearly with the number of functions to be handled. Eventually, the decryption key could be distributed among several on-line servers that would help the user to get $\boldsymbol{f}(\boldsymbol{m})$. But the decryption key (even distributed) is critical, and using it too often is quite dangerous for its privacy. Functional encryption suffers neither of those drawbacks, since the master key is used for issuing the functional keys only.

Functional Encryption. While a series of papers proposed secure constructions of FE schemes for general circuits [GGH$^+$13,BCP14,Wat15,GGHZ16], they are very inefficient, perhaps because of their generality. Some concrete constructions, that only target specific functions of interest have recently been proposed. Notably, the construction in [ABDP15] on the inner-product seems very interesting in a support vector machine setup in machine learning (see below), as both the learning and the classifying phases can be handled by computing several inner-products. Some improvements have thereafter been presented to achieve adaptive security and/or function hiding [BJK15, ALS15, ALS16, ABDP16].

While this functionality has found many applications, it still suffers from the classical drawback of functional encryption: a combination of too many functional keys leads to a total leakage of the plaintext. This is obviously the case for an inner-product functionality, as an adversary yielding ℓ keys for a vector-plaintext of length ℓ simply has to invert an $\ell \times \ell$ matrix to recover the entire vector-plaintext. We stress that it is not a flaw of any of the above constructions, but a direct consequence of the functionality itself: being able to compute $\boldsymbol{f}(\boldsymbol{m})$ on any ciphertext, and for too many functionalities $\boldsymbol{f}$, one can get the whole database in clear.

Machine Learning. Machine-Learning is a subfield of computer-science that allows pattern recognition [JDM00] on an automated basis independent of the specificities of the actual data. In particular, supervised learning is where a set of data is first manually labeled and learned-on, then a classifying algorithm can be used to assign a label to new data. A particular well-studied set of supervised learning algorithms are the support vector machine (SVM) algorithms [HDO$^+$98, SS02], that take a geometrical approach and try to determine linear classifiers ($n - 1$ dimension hyperplane) to separate two sets of points. They notably only make use of inner-products for both the learning phase (finding the optimal hyperplane) and the classifying phase (assigning the correct value to new data, in which case a fixed vector is used).

These techniques are widely used nowadays, and nobody could miss them anymore, as they are of great help for making researches on big data (data mining) or for taking decisions. But this is at the price of a big loss of privacy, since huge unencrypted databases are used, for both the learning and the classifying tasks. Being able to address these tasks on encrypted databases is the holy grail!

This works thus focuses on this problem, and does so by introducing an additional player to the classical FE setup: the *helper*, to limit and control the leakage of information. This means that our new construction will be interactive, but this fits right with the new *cloud* paradigm, where everything is online anyway.

Functional Encryption with Helper. In order to control and limit the information learned by the users, an interaction with this helper is required in order to get anything about the encrypted database, in addition to the functional decryption keys.

More precisely, a functional decryption key is still a first requirement to start the decryption process, but an interaction with the helper is another requirement, to conclude the process. The role of this helper is to apply a restriction policy, to limit the number of decryption queries the users can ask, so that the privacy of the global database cannot be endangered.

On the other hand, it cannot be trusted: it should not be able to get any information about the plaintext, and neither should he learn any information about the queries asked by the user. It should be an *oblivious* helper, that helps the user, without learning anything other than the minimum information needed to apply the restriction policy (e.g. a query was made on a given ciphertext). This is in contrast with the simpler approach mentioned above, with on-line servers that help to compute $\boldsymbol{f}(\boldsymbol{m})$: either they learn everything (the plaintext and the function to be applied) or secure multi-party computation is required.

We further restrict this functional encryption scheme by allowing a different primitive than the classical decryption: the *decryption test*, in which the user provides in advance a set V of possible values for $\boldsymbol{f}(\boldsymbol{m})$ and, with one interaction with the helper, only learns whether the actual value is in the set. For the inner-product case, we provide an efficient construction for the test of membership of an interval: the user can learn whether the inner-product value is in an interval of his choice, but does not get more; the helper just learns the ciphertext and the size of the interval, which are important information to evaluate the global leakage on the database (the larger is the interval, the less information is leaked).

Our functional encryption scheme with helper is akin to combining classical functional encryption with a secure two-party computation protocol in which one party (the helper) provides its decryption key as input and only learns the ciphertext used, but neither the result nor the function, and the other party (the user) provides the ciphertext and the function as input and only learn $\boldsymbol{f}(\boldsymbol{m})$, and nothing else. However, the scheme we propose is much more efficient than such generic compositions: we tailor our construction to our specific problem.

Organization. The next section defines the formal model for such a functional encryption scheme. Then we will briefly introduce some of the key cryptographic building blocks that our concrete protocol will require. Lastly, we present our construction and prove its security in our security model.

2 Functional Encryption Model

We now define the formal model for our new functional encryption scheme with oblivious helper, that generalizes regular functional encryption schemes by adding a player (the *oblivious helper*) and introducing a *decryption test* algorithm, that reveals whether $\boldsymbol{f}(\boldsymbol{m}) \in V$ for a specified set of values V, instead of a regular decryption that reveals $\boldsymbol{f}(\boldsymbol{m})$.

This section first formalizes our new functional encryption with helper, with its correctness and then defines the security model to encompass our expected security goals.

2.1 Description

In order to limit the leakage of information to the users, we define a new functional encryption scheme with the same properties as usual, but which requires helper queries for any decryption, or even just decryption test (for a ciphertext C of a message $\boldsymbol{m}$, does $\boldsymbol{f}(\boldsymbol{m})$ belong to V?).

- $\mathsf{Setup}(1^\lambda)$: Given the security parameter λ, it generates the global parameters param of the system;
- $\mathsf{KeyGen}_e(\mathsf{param})$: Given the global parameters param, it generates the master secret key msk, the encryption key ek and the verification key vk. This encryption key is used in the encryption of a message while the verification key allows to check the signature of the functional secret keys;
- $\mathsf{KeyGen}_h(\mathsf{param})$ Given the global parameters param, it generates the helper public and secret keys, $\mathsf{pk}\mathcal{H}$ and $\mathsf{sk}\mathcal{H}$;
- $\mathsf{KeyDer}(\mathsf{msk}, \boldsymbol{f})$: Given the master secret key msk and a function $\boldsymbol{f}$, it generates the functional secret key $\mathsf{sk}\boldsymbol{f}$;
- $\mathsf{Encrypt}(\mathsf{ek}, \mathsf{pk}\mathcal{H}, \boldsymbol{m})$: Given the encryption key ek, the helper public key $\mathsf{pk}\mathcal{H}$, and a message $\boldsymbol{m}$, it generates a ciphertext C;
- $\mathsf{TestDecrypt}^{\mathcal{H}}(\mathsf{pk}\mathcal{H}, C, \mathsf{sk}\boldsymbol{f}, V)$: Given a ciphertext C, the functional secret key $\mathsf{sk}\boldsymbol{f}$, a set V of possible values for the evaluation of $\boldsymbol{f}$ on the plaintext and the helper public key $\mathsf{pk}\mathcal{H}$, it confirms (with output 1) or not (with output 0) that this evaluation of $\boldsymbol{f}$ on the plaintext belongs to V, with an interaction with the test-helper $\mathcal{H}$ owning $\mathsf{sk}\mathcal{H}$.

For *correctness* we require that:

$$\begin{aligned} &\forall \mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda),\\ &\forall (\mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}_e(\mathsf{param}),\\ &\forall \mathsf{pk}\mathcal{H} \leftarrow \mathsf{KeyGen}_h(\mathsf{param}),\\ &\forall (\boldsymbol{m} \in \mathcal{M}, \boldsymbol{f} \in \mathcal{F}, V),\\ &\forall C \leftarrow \mathsf{Encrypt}(\mathsf{ek}, \mathsf{pk}\mathcal{H}, \boldsymbol{m}),\\ &\forall \mathsf{sk}\boldsymbol{f} \leftarrow \mathsf{KeyDer}(\mathsf{msk}, \boldsymbol{f}),\\ \mathsf{TestDecrypt}^{\mathcal{H}}(\mathsf{pk}\mathcal{H}, C, \mathsf{sk}\boldsymbol{f}, V) = 1 \iff &\boldsymbol{f}(\boldsymbol{m}) \in V \end{aligned}$$

For efficiency reasons, we will focus on schemes that have one-round $\mathsf{TestDecrypt}$ protocols. This means that the user makes one interaction only with $\mathcal{H}$ during the $\mathsf{TestDecrypt}$ protocol. The helper might output $\perp$ in case of bad inputs, and then $\mathsf{TestDecrypt}$ aborts. KeyDer and $\mathsf{Encrypt}$ might output $\perp$ in case of invalid inputs.

2.2 Security Model

First, our main contribution is on the limit of the leakage of information one can get from the functional keys, thanks to the helper queries: while the functional keys give some decryption rights to the user, the helper will restrict the quantity of learned information (*e.g.*, limiting the number of $\mathsf{TestDecrypt}$-queries). After such $\mathsf{TestDecrypt}$-queries, an adversary should not learn more than whether $\boldsymbol{f}(\boldsymbol{m}) \in V$ or not for the plaintext $\boldsymbol{m}$ encrypted in the ciphertext C, and of course nothing about other plaintexts (indistinguishability).

On the other hand, in the case of collusion with the helper $\mathcal{H}$ owning $\mathsf{sk}\mathcal{H}$, no-one should learn more than the evaluation of the functions for which it got the functional secret keys on the plaintexts. This is essentially the classical notion of semantic security for basic functional encryption schemes, that we can fall-back on if the helper was to be compromised (under such a collusion assumption, this is the best one can achieve). We call it *data privacy* in our setting.

Moreover, the helper $\mathcal{H}$ should not learn more information than required to restrict the amount of information the user can get: its role is to limit the number of queries, but it should not be able to get information on any plaintexts (which is already guaranteed by the above data privacy security notion, since $\mathsf{sk}\mathcal{H}$ is known to the adversary) and it should not learn anything about the queries (query privacy). There may however be several limitations: the helper $\mathcal{H}$ may limit the number of queries globally, by a given user[1] whatever the queries are; or the number of queries can be limited per ciphertext (globally or per user). In the latter case, we allow the helper to know on which ciphertext C the $\mathsf{TestDecrypt}$-query is asked, which looks important to evaluate the amount of information the user can get about a specific ciphertext.

In any case, it is important that the helper cannot provide a wrong answer to the user (verifiability), but still, it should not learn the answer either.

We now formally define the experiments for those four security notions, where the $\mathsf{Initialize}$-query must be called first, and once only, while the $\mathsf{Finalize}$-query must be called last, and once only. All other queries can be called as many times as necessary and in any order.

Indistinguishability (`IND`). This notion is the most important contribution in our security model: it models the restrictions for the user with $\mathsf{TestDecrypt}$-queries. The adversary plays the role of a user, and everything that it can learn with the system could be learned with explicit queries to $\mathsf{Test}(C, \boldsymbol{f}, V)$, on legitimate ciphertexts $C = \mathsf{Encrypt}(\boldsymbol{m})$ and legitimate functions $\boldsymbol{f}$ asked to KeyDer, that simply checks whether $\boldsymbol{f}(\boldsymbol{m}) \in V$ or not. More precisely, in the experiment below, we expect the AskH-queries to be answered using the above Test-queries only:

- $\mathsf{Initialize}(\lambda)$: it runs $\mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda)$, generates the keys $(\mathsf{msk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}_e(\mathsf{param})$, $(\mathsf{pk}\mathcal{H}, \mathsf{sk}\mathcal{H}) \leftarrow \mathsf{KeyGen}_o(\mathsf{param})$, and outputs $(\mathsf{param}, \mathsf{ek}, \mathsf{pk}\mathcal{H})$;
- $\mathsf{KeyDer}(\boldsymbol{f})$: it stores $\boldsymbol{f}$ in the list Ψ and outputs $\mathsf{sk}\boldsymbol{f} \leftarrow \mathsf{KeyDer}(\mathsf{msk}, \boldsymbol{f})$;
- $\mathsf{Encrypt}(\boldsymbol{m})$: it computes $C \leftarrow \mathsf{Encrypt}(\mathsf{ek}, \mathsf{pk}\mathcal{H}, \boldsymbol{m}^b)$, stores C in the list Φ, and outputs C;
- $\mathsf{AskH}(M)$: if $b = 0$, it runs the $\mathsf{TestDecrypt}$ procedure with $\mathcal{H}$ (knowing $\mathsf{sk}\mathcal{H}$), by sending M; otherwise it runs it with the simulator $\mathcal{S}$ that only has access to Test, and forwards the answer;
- $\mathsf{Finalize}(b')$: Returns $(b = b')$.

The adversary wins if $\mathsf{Finalize}$ outputs `true`. A good adversary is an adversary that wins significantly more often than half of the time against any simulator $\mathcal{S}$:

$$\mathsf{Adv}^{\mathtt{ind}}_{\mathcal{S}}(\mathcal{A}) = \Pr[\mathsf{Finalize} = \mathtt{true}] - 1/2 \geq \varepsilon,$$

for some non-negligible ε.

We stress that for AskH-queries answered by $\mathcal{S}$, the latter has just access to Ψ, Φ, and helper $\mathsf{Test}(C, \boldsymbol{f}, V)$ that answers whether $\boldsymbol{f}(\boldsymbol{m}) \in V$ or not, for legitimate ciphertexts $C \in \Phi$ and functions $\boldsymbol{f} \in \Psi$.

Data Privacy (`D-Priv`). This security notion is actually the classical semantic security for basic functional encryption schemes, where the adversary should distinguish the ciphertexts of two plaintexts of its choice, with the constraint that the functions for which it gets the functional secret keys should evaluate the same way on the two plaintexts. We stress that in this experiment, the adversary controls both a player and the helper $\mathcal{H}$. It is thus given the helper secret key $\mathsf{sk}\mathcal{H}$. The challenger flips a random coin $b \overset{\$}{\leftarrow} \{0, 1\}$, which is used in the following helpers:

- $\mathsf{Initialize}(\lambda)$: it runs $\mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda)$, generates the keys $(\mathsf{msk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}_e(\mathsf{param})$, $(\mathsf{pk}\mathcal{H}, \mathsf{sk}\mathcal{H}) \leftarrow \mathsf{KeyGen}_o(\mathsf{param})$, and outputs $(\mathsf{param}, \mathsf{ek}, \mathsf{sk}\mathcal{H}, \mathsf{pk}\mathcal{H})$;
- $\mathsf{KeyDer}(\boldsymbol{f})$: it stores $\boldsymbol{f}$ in the list Ψ and outputs $\mathsf{sk}\boldsymbol{f} \leftarrow \mathsf{KeyDer}(\mathsf{msk}, \boldsymbol{f})$;

[1] This of course means that such queries occur on an authenticated channel for the user and that in case of collusion between several users, their number of authorized query sum up

- $\mathsf{LREncrypt}(\boldsymbol{m}^0, \boldsymbol{m}^1)$: it stores $(\boldsymbol{m}^0, \boldsymbol{m}^1)$ in the list Φ and outputs the ciphertext $C \leftarrow \mathsf{Encrypt}(\mathsf{ek}, \mathsf{pk}\mathcal{H}, \boldsymbol{m}^b)$;
- $\mathsf{Finalize}(b')$: if *fresh*, it returns $(b = b')$. Otherwise it returns `false`.

The adversary wins if Finalize outputs `true`. A good adversary is an adversary that wins significantly more often than half the time:

$$\mathsf{Adv}^{\texttt{d-priv}}(\mathcal{A}) = \Pr[\mathsf{Finalize} = \texttt{true}] - 1/2 \geq \varepsilon,$$

for some non-negligible ε. We define the freshness for the data privacy experiment, in order to exclude trivial attacks:

Definition 1 (Freshness). Data privacy experiments are considered *fresh* if for any $\boldsymbol{f} \in \Psi$ and any $(\boldsymbol{m}^0, \boldsymbol{m}^1) \in \Phi$, $\boldsymbol{f}(\boldsymbol{m}^0) = \boldsymbol{f}(\boldsymbol{m}^1)$.

Query Privacy (`Q-Priv`). In this experiment, the adversary still plays the role of the helper $\mathcal{H}$, and is thus given the helper secret key $\mathsf{sk}\mathcal{H}$, but its aim is at understanding users' helper-queries. One thus assumes the user can get any functional key of its choice, but helper-queries can be asked on legitimate ciphertexts only, hence the following available queries:

- $\mathsf{Initialize}(\lambda)$: it runs $\mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda)$, generates the keys $(\mathsf{msk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}_e(\mathsf{param})$, $(\mathsf{pk}\mathcal{H}, \mathsf{sk}\mathcal{H}) \leftarrow \mathsf{KeyGen}_o(\mathsf{param})$, and outputs $(\mathsf{param}, \mathsf{ek}, \mathsf{sk}\mathcal{H}, \mathsf{pk}\mathcal{H})$;
- $\mathsf{Encrypt}(\boldsymbol{m})$: it computes $C \leftarrow \mathsf{Encrypt}(\mathsf{ek}, \mathsf{pk}\mathcal{H}, \boldsymbol{m})$ and stores C in Λ. Then it outputs C;
- $\mathsf{LRTestDecrypt}(C, (\boldsymbol{f}^0, V^0), (\boldsymbol{f}^1, V^1))$:
  - If $C \notin \Lambda$, it outputs $\perp$.
  - Otherwise, it computes $\mathsf{sk}\boldsymbol{f}^b \leftarrow \mathsf{KeyDer}(\mathsf{msk}, \boldsymbol{f}^b)$, and runs $\mathsf{TestDecrypt}(C, \mathsf{sk}\boldsymbol{f}^b, V^b)$. Any interaction with $\mathcal{H}$ in this procedure is done with $\mathcal{A}$ instead;
- $\mathsf{Finalize}(b')$: Returns $(b = b')$.

As above, we define

$$\mathsf{Adv}^{\texttt{q-priv}}(\mathcal{A}) = \Pr[\mathsf{Finalize} = \texttt{true}] - 1/2.$$

One can note that the above security game requires that no information about the outcome of TestDecrypt is learned by the helper $\mathcal{H}$, even being malicious. Indeed, since it knows $\boldsymbol{m}$ and $(\boldsymbol{f}^0, \boldsymbol{v}^0), (\boldsymbol{f}^1, \boldsymbol{v}^1)$, if $(\boldsymbol{f}^0(\boldsymbol{m}) \in V^0) \neq (\boldsymbol{f}^1(\boldsymbol{m}) \in V^1)$, the outcome of the TestDecrypt would reveal b.

Verifiability (`SND`). The last security notion is more a soundness property: any misbehavior should be detectable by the user. More precisely, both the functional keys and the decryption tests (unless the helper outputs $\perp$) should be verifiable. This means that the probability that the user accepts a false result is negligible.

3 Building Blocks

For our construction, several classical cryptographic tools will be useful, and namely to guarantee verifiability and privacy. It is the purpose of this section to introduce them.

First, we introduce the well-known ElGamal encryption scheme [ElG84], whose homomorphic property is at the basis of the Inner-Product functional encryption scheme [ABDP15, ABDP16, ALS15, ALS16] on which this work builds on.

The interactive protocol between the holder of the functional key $\mathsf{sk}\boldsymbol{f}$ and the helper $\mathcal{H}$ will require blindly relating a decryption query to a ciphertext (so as to implement a policy on the number of operations on a ciphertext). Hence, Non-Interactive Zero-Knowledge Proofs are presented next. They allow to guarantee that the values where honestly computed, without leaking the secrets involved.

Eventually, the functional keys themselves need to be authenticated, but without any way, for the helper, to learn more than whether it is legitimate or not. More precisely, it should not be able to link multiple usages of a single key or to relate a presented key to one delivered by the holder of the master secret key. This can be achieved through a special kind of signature schemes: randomizable signature schemes.

3.1 ElGamal Encryption Scheme

We will use the famous ElGamal encryption [ElG84], in order to commit and/or encrypt messages in a cyclic group $\mathbb{G}$ of prime order p with generator g. First, one generates the keys $\mathsf{dk} \xleftarrow{\$} \mathbb{Z}_p$ and $\mathsf{ek} \leftarrow g^{\mathsf{dk}}$. To encrypt a message $m \in \mathbb{G}$, one samples $r \xleftarrow{\$} \mathbb{Z}_p$ and computes $C = (c_0 \leftarrow g^r, c_1 \leftarrow m \cdot \mathsf{ek}^r)$. To decypt, the receiver simply computes $m \leftarrow c_1 / c_0^{\mathsf{dk}}$.

This scheme achieves the $\mathtt{IND-CPA}$ security level under the DDH assumption in $\mathbb{G}$ [BDPR98].

This encryption scheme is multiplicatively homomorphic in $\mathbb{G}$: if one multiplies component-wise, the encryption of M and the encryption of M', one gets the encryption of $M \cdot M'$. One often needs additive homomorphism in $\mathbb{Z}_p$. In such a case, one encrypts g^m. But then, when decrypting as explained above, one just gets g^m, and so one has to compute a discrete logarithm to get back the plaintext $m \in \mathbb{Z}_p$. Hence, one has to limit the ciphertexts to encrypt small messages $m \in \mathcal{M}$.

3.2 Non-Interactive Zero-Knowledge Proofs

In order to prove the wellformedness of the queries and the answers, the players will have to prove their honest behavior. It will be sufficient for our purpose to prove some relationships between hidden scalars, as exponents in possibly different groups $\mathbb{G}_j$ that all share the same prime order p. More precisely, a player will have to prove the existence of scalars $(x_i)_i$ in $\mathbb{Z}_p$, so that for group elements $G_{i,j}, T_j \in \mathbb{G}_j$, for all j, $T_j = \prod_i G_{i,j}^{x_i}$.

As explained in [CS97], this can be proven using Schnorr-like proofs [Sch90a, Sch91]: for each i, the prover samples $k_i \xleftarrow{\$} \mathbb{Z}_p$, and sets $R_j = \prod_i G_{i,j}^{k_i}$; upon receiving a random challenge $e \xleftarrow{\$} \mathbb{Z}_p$, the prover computes $s_i = k_i - ex_i \mod p$ and sends them back to the verifier. The latter can then check that, for all j, $R_j = T_j^e \times \prod_i G_{i,j}^{s_i}$.

By using the Fiat-Shamir heuristic in the random oracle model [BR93], proven in [PS96, PS00], we can turn it into a non-interactive zero-knowledge proof: the prover samples $k_i \xleftarrow{\$} \mathbb{Z}_p$, and sets $R_j = \prod_i G_{i,j}^{k_i}$; it then computes itself the challenge $e = \mathcal{H}((R_j)_j)$ and the answers $s_i = k_i - ex_i \mod p$. It sends back the full proof: $\mathit{\Pi} = (e, (s_i)_i)$, which can be checked by $e \stackrel{?}{=} \mathcal{H}((T_j^e \times \prod_i G_{i,j}^{s_i})_i)$.

We stress that we are dealing with proofs of membership only, and not proofs on knowledge: the rewind for the forking lemma [PS96] is only used for the proof of the soundness, but not for extraction.

About the complexity, one can note that the size of the proof is just linear in the number of scalars, and independent of the number of relations: if there are n scalars and m relations, the size of the proof is just $n+1$ scalars (the challenge e, and the answers $(s_i)_i$).

3.3 Randomizable Signatures

In addition, in order to limit queries to legitimate ones, the functional keys will be signed. But they should not be revealed during queries, just proven legitimate. This is in the same vein as anonymous credentials [CL04]. For better efficiency, we will use the signature scheme recently proposed [PS16]: in order to sign ℓ-scalar messages, one defines a bilinear group setting $(p, \mathbb{G}, g, \tilde{\mathbb{G}}, \tilde{g}, e) \xleftarrow{\$} \mathsf{GroupGen}(1^\lambda)$. The signer chooses $\ell+1$ scalars $x, y_1, \ldots, y_\ell \xleftarrow{\$} \mathbb{Z}_p$, and sets the signing key as $\mathsf{sk} \leftarrow (x, y_1, \ldots, y_\ell)$ and the verification $\mathsf{vk} \leftarrow \left(\tilde{X} = \tilde{g}^x, \tilde{Y}_1 = \tilde{g}^{y_1}, \ldots, \tilde{Y}_\ell = \tilde{g}^{y_\ell}\right)$. The signature of $(m_i)_i$ is the pair $\sigma = (\sigma_1 \leftarrow h, \sigma_2 \leftarrow h^{x+\sum_i m_i y_i})$, for a randomly chosen $h \xleftarrow{\$} \mathbb{G} \backslash \{1_\mathbb{G}\}$. This signature σ on the tuple $(m_i)_i$ can be checked by

$$\sigma_1 \neq 1_\mathbb{G} \quad \text{and} \quad e(\sigma_1, \tilde{X}) \cdot \prod_{i=1}^{i=\ell} e(\sigma_1, \tilde{Y}_i)^{m_i} = e(\sigma_2, \tilde{g}).$$

It is important to note that, for any random $r \xleftarrow{\$} \mathbb{Z}_p$, $\sigma' \leftarrow (\sigma_1^r, \sigma_2^r)$ is another valid signature on $(m_i)_i$.

This scheme achieves the EUF−CMA security level under a derived assumption of LRSW [LRSW99], defined in [PS16]. It has been proven in the asymmetric bilinear generic group model. We stress that type 3 pairings, where the XDH assumption holds are required (see below).

4 Our Protocol

We first define our functional encryption with oblivious helper scheme, which is a natural extension of [ABDP16, ALS15, ALS16], using the signature from [PS16] in order to certify functional keys. The latter makes use of a pairing-friendly group of type 3, where the XDH assumption holds, and even more.

We will first start by a quick review of the computational assumptions needed, followed by the description of the protocol. For the sake of simplicity, we at first limit ourselves to TestDecrypt-queries on singletons: $V = \{v\}$. We then provide the security analysis. Then, in section 4.6, we show how to extend the TestDecrypt-queries to intervals: $V = \{v, \ldots, v + L - 1\}$, while just leaking the length L.

4.1 Computational Assumptions

Bilinear groups are a set of three cyclic groups $\mathbb{G}$, $\tilde{\mathbb{G}}$, and $\mathbb{G}_T$ of prime order p along with a bilinear map $e : \mathbb{G} \times \tilde{\mathbb{G}} \to \mathbb{G}_T$ with the following properties:

1. for all $g \in \mathbb{G}, \tilde{g} \in \tilde{\mathbb{G}}$ and $a, b \in \mathbb{Z}_p$, $e(g^a, \tilde{g}^b) = e(g, \tilde{g})^{a \cdot b}$;
2. for $g \neq 1_{\mathbb{G}}$ and $\tilde{g} \neq 1_{\tilde{\mathbb{G}}}$, $e(g, \tilde{g}) \neq 1_{\mathbb{G}_T}$;
3. the map e is efficiently computable.

Galbraith, Paterson, and Smart [GPS08] defined three types of pairings: in type 1, $\mathbb{G} = \tilde{\mathbb{G}}$; in type 2, $\mathbb{G} \neq \tilde{\mathbb{G}}$ but there exists an efficient homomorphism $\phi : \tilde{\mathbb{G}} \to \mathbb{G}$, while no efficient one exists in the other direction; in type 3, $\mathbb{G} \neq \tilde{\mathbb{G}}$ and no efficiently computable homomorphism exists between $\mathbb{G}$ and $\tilde{\mathbb{G}}$, in either direction.

Recent cryptosystems focus on type 3 pairings for their efficiency. In addition, they allow the Decision Diffie-Hellman assumption to hold in either $\mathbb{G}$ or $\tilde{\mathbb{G}}$ (which is also known as the XDH assumption), which is not the case for type 1 pairings. As for the Pointcheval-Sanders signature, we stress that using type 1 or type 2 pairings would make both the signature and the use of the El Gamal encryption totally insecure.

We denote $(p, \mathbb{G}, g, \tilde{\mathbb{G}}, \tilde{g}, e) \xleftarrow{\$} \mathsf{GroupGen}(1^\lambda)$ the generation of such a type 3 pairing setting. In addition, we assume hardness of the decisional Diffie-Hellman problem in the group $\mathbb{G}$: given $X = g^x$, $Y = g^y$ and $Z = g^z$, where z is either a random scalar or $xy \bmod p$, no efficient adversary can distinguish the two cases with significant advantage. This allows us to later use the ElGamal encryption in this group $\mathbb{G}$: the DDH assumption in $\mathbb{G}$ provides the semantic security (indistinguishability).

4.2 Description

We present our construction, with helper queries for the TestDecrypt. It is based on the additively homomorphic variant of ElGamal. Since the values of interest are $\boldsymbol{f} \cdot \boldsymbol{m}$, where $\boldsymbol{m} \in \mathbb{Z}_p^\ell$ are the encrypted messages and $\boldsymbol{f} \in \mathbb{Z}_p^\ell$ the linear combinations, to allow decryption these values have to be small enough to be obtained by a discrete logarithm computation in reasonable time Time. Hence, we define $\mathcal{M} \subset \mathbb{Z}_p^\ell$ and $\mathcal{F} \subset \mathbb{Z}_p^\ell$ the admissible message space and function space so that for any $\boldsymbol{m} \in \mathcal{M}$ and any $\boldsymbol{f} \in \mathcal{F}$, one can compute the discrete logarithm of $g^{\boldsymbol{f} \cdot \boldsymbol{m}}$ in time less than Time.

- $\mathsf{Setup}(1^\lambda)$: it samples $(p, \mathbb{G}, g, \tilde{\mathbb{G}}, \tilde{g}, e) \xleftarrow{\$} \mathsf{GroupGen}(1^\lambda)$ as well as independent generators $g_s, g_t, g_\tau \xleftarrow{\$} \mathbb{G}$. It sets $\mathsf{param} = (\mathbb{G}, \tilde{\mathbb{G}}, p, g, g_\tau, g_s, g_t, \tilde{g}, e)$;
- $\mathsf{KeyGen}_e(\mathsf{param})$: it samples $\boldsymbol{s} = (s_1, \ldots, s_\ell)$ and $\boldsymbol{t} = (t_1, \ldots, t_\ell) \xleftarrow{\$} \mathbb{Z}_p^\ell$, and sets $\mathsf{ek} = \boldsymbol{h}$, with $h_i = g_s^{s_i} g_t^{t_i}$, for $i = 1, \ldots, \ell$, it also samples $x, y_1, \ldots, y_\ell \xleftarrow{\$} \mathbb{Z}_p$, and sets $\mathsf{vk} = (\tilde{X} = \tilde{g}^x, \tilde{Y}_1 = \tilde{g}^{y_1}, \ldots, \tilde{Y}_\ell = \tilde{g}^{y_\ell})$ and $\mathsf{msk} = (\boldsymbol{s}, \boldsymbol{t}, x, y_1, \ldots, y_\ell)$;
- $\mathsf{KeyGen}_o(\mathsf{param})$: it samples $\mathsf{sk}\mathcal{H} \xleftarrow{\$} \mathbb{Z}_p$ and sets $\mathsf{pk}\mathcal{H} = g^{\mathsf{sk}\mathcal{H}}$;
- $\mathsf{KeyDer}(\mathsf{msk}, \boldsymbol{f})$: for a vector $\boldsymbol{f} = (f_1, \ldots, f_\ell) \in \mathcal{F}$, it samples $h \xleftarrow{\$} \mathbb{G}$, and outputs $\mathsf{sk}\boldsymbol{f} = (\mathsf{sk}\boldsymbol{f}_0 = h, \mathsf{sk}\boldsymbol{f}_1 = \boldsymbol{f} \cdot \boldsymbol{s} \bmod p, \mathsf{sk}\boldsymbol{f}_2 = \boldsymbol{f} \cdot \boldsymbol{t} \bmod p, \mathsf{sk}\boldsymbol{f}_3 = h^{x + \sum_i y_i f_i})$. One can check the validity of this key, since $\prod_i h_i^{f_i} = g_s^{\mathsf{sk}\boldsymbol{f}_1} \cdot g_t^{\mathsf{sk}\boldsymbol{f}_2}$ and $e(\mathsf{sk}\boldsymbol{f}_0, \tilde{X} \prod_i \tilde{Y}_i^{f_i}) = e(\mathsf{sk}\boldsymbol{f}_3, \tilde{g})$;
- $\mathsf{Encrypt}(\mathsf{ek}, \mathsf{pk}\mathcal{H}, \boldsymbol{m} = (m_1, \ldots, m_\ell) \in \mathcal{M})$: it samples $r \xleftarrow{\$} \mathbb{Z}_p$ and $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_\ell) \xleftarrow{\$} \mathbb{Z}_p^\ell$, and outputs $C = (C_0, C_0', C_1, D_1, \ldots, C_\ell, D_\ell)$, where $C_0 = g_s^r$, $C_0' = g_t^r$, and $C_i = \mathsf{pk}\mathcal{H}^{\alpha_i} g^{m_i} h_i^r$, $D_i = g^{\alpha_i}$;
- $\mathsf{TestDecrypt}(\mathsf{pk}\mathcal{H}, C, \mathsf{sk}\boldsymbol{f}, (v))$: it computes

  $$A = \prod\nolimits_i C_i^{f_i} / (C_0^{\mathsf{sk}\boldsymbol{f}_1} \cdot C_0'^{\,\mathsf{sk}\boldsymbol{f}_2}) \quad B = \prod\nolimits_i D_i^{f_i}.$$

  One can note that $A = \mathsf{pk}\mathcal{H}^{\boldsymbol{f} \cdot \boldsymbol{\alpha}} \cdot g^{\boldsymbol{f} \cdot \boldsymbol{m}}$ and $B = g^{\boldsymbol{f} \cdot \boldsymbol{\alpha}}$, which is an ElGamal encryption of g^v under the public key $\mathsf{pk}\mathcal{H}$ if and only if $v = \boldsymbol{f} \cdot \boldsymbol{m} \bmod p$. This is the verification the user asks to the helper $\mathcal{H}$ (which knows the decryption key $\mathsf{sk}\mathcal{H}$), through the interactive protocol described below. Note that this process will not leak any information to $\mathcal{H}$ apart from the ciphertext used C.

Interactions with the helper
Depending on the limitations the helper $\mathcal{H}$ might want to apply, the user might have to partially specify the query. As explained above, while it looks quite important to hide $\boldsymbol{f}$ and v, the helper might want to be able to count the number of queries on each specific ciphertext. As a consequence, we focus on a description of $\mathcal{H}$ that explicitly receives the ciphertext C (which has to be a legitimate one: it is either signed by the owner of the database, or the database is publicly available), but not $\boldsymbol{f}$ nor v. Instead, the user commits on them, and runs an equality test with the helper. It thus computes

- a randomized signature $W_0 \leftarrow \mathsf{sk}\boldsymbol{f}_0^\sigma$, $W_1 \leftarrow \mathsf{sk}\boldsymbol{f}_3^\sigma$, of the function vector $\boldsymbol{f}$, for a random $\sigma \xleftarrow{\$} \mathbb{Z}_p$;
- an ElGamal encryption (extractable commitment) of W_1 under the key g_τ, $U_1 \leftarrow g^{\rho_1}$, $T_1 \leftarrow W_1 \cdot g_\tau^{\rho_1}$, for random $\rho_1 \xleftarrow{\$} \mathbb{Z}_p$;
- an ElGamal encryption (extractable commitment) of g^v under the key g_τ, $U_2 \leftarrow g^{\rho_2}$, $T_2 \leftarrow g^v \cdot g_\tau^{\rho_2}$, for random $\rho_2 \xleftarrow{\$} \mathbb{Z}_p$;
- a re-randomization of (B, A): $U' \leftarrow B \cdot g^{\rho'}$, $T' \leftarrow A \cdot Z \cdot \mathsf{pk}\mathcal{H}^{\rho'} / g^v$, for random $\rho' \xleftarrow{\$} \mathbb{Z}_p$ and $Z \xleftarrow{\$} \mathbb{G}$. The pair (U', T') is a re-randomization of the ElGamal ciphertext (B, A) of $g^{\boldsymbol{f} \cdot \boldsymbol{m}}$ under the key $\mathsf{pk}\mathcal{H}$, into an encryption of $Z \cdot g^{\boldsymbol{f} \cdot \boldsymbol{m} - v}$, and thus of Z if and only if $\boldsymbol{f} \cdot \boldsymbol{m} = v \bmod p$;
- an ElGamal encryption of Z under a fresh key $h' = g^\beta$: $U \leftarrow g^\rho$, $T \leftarrow Z \cdot h'^\rho$, for random $\beta, \rho \xleftarrow{\$} \mathbb{Z}_p$;

as well as a proof of validity: existence of ρ, ρ', ρ_1, ρ_2, $\mathsf{sk}\boldsymbol{f}_1$, $\mathsf{sk}\boldsymbol{f}_2$, v, and $f_1, \ldots, f_\ell$ such that

$$g^{\rho_1} = U_1 \qquad \prod_i e(W_0, \tilde{Y}_1)^{f_i} e(g_\tau, \tilde{g})^{\rho_1} = e(T_1, \tilde{g}) / e(W_0, \tilde{X}) \tag{1}$$

$$g^{\rho_2} = U_2 \qquad g_\tau^{\rho_2} \cdot g^v = T_2 \tag{2}$$

$$g_s^{-\mathsf{sk}\boldsymbol{f}_1} \cdot g_t^{-\mathsf{sk}\boldsymbol{f}_2} \cdot \prod_i h_i^{f_i} = 1_{\mathbb{G}} \tag{3}$$

$$\begin{gathered} g^{\rho'} \cdot \prod_i D_i^{f_i} = U', g^{\rho} = U \\ \mathsf{pk}\mathcal{H}^{-\rho'} \cdot h'^{\rho} \cdot C_0^{\mathsf{sk}\boldsymbol{f}_1} C_0'^{\mathsf{sk}\boldsymbol{f}_2} \cdot g^v \cdot \prod_i C_i^{-f_i} = T/T'. \end{gathered} \tag{4}$$

Note that the line (1), along with the fact that $W_0 \neq 1_{\mathbb{G}}$, shows that (U_1, T_1) commits to the second part (together with W_0) of a valid signature of the function vector $\boldsymbol{f}$, that satisfies the relations below, under $g_\mathcal{T}$. The line (2) shows that (U_2, T_2) commits to g^v, that satisfies the relations below, under $g_\mathcal{T}$. The line (3) shows that $(\mathsf{sk}\boldsymbol{f}_1, \mathsf{sk}\boldsymbol{f}_2)$ is the functional decryption key of $\boldsymbol{f}$. The line (4) proves that if (U, T) encrypts some Z under h', then (U', T') encrypts $Z \cdot g^{\boldsymbol{f}\cdot\boldsymbol{m}-v}$ under $\mathsf{pk}\mathcal{H}$.

Using a Schnorr-like proof of knowledge [Sch90b], the user sends 10 group elements (W_0, U_1, T_1), (U_2, T_2), (h', U, T), and (U', T'), as well as $\ell + 9$ scalars for the proof (an id for the ciphertext C, the proof challenge and the answers for the $\ell + 7$ witnesses ρ, ρ', ρ_1, ρ_2, $\mathsf{sk}\boldsymbol{f}_1$, $\mathsf{sk}\boldsymbol{f}_2$, v, and $f_1, \ldots, f_\ell$).

After having checked the proofs, the helper $\mathcal{H}$ checks if the relevant policy (number of queries on C by the user) is violated, in which case it aborts by returning $\perp$. Otherwise, it computes $V = T'/U'^{\mathsf{sk}\mathcal{H}}$, which is equal to $Z \cdot g^{\boldsymbol{f}\cdot\boldsymbol{m}-v}$, and thus to Z, if and only of $\boldsymbol{f} \cdot \boldsymbol{m} = v \bmod p$. It now completes the comparison protocol, by rerandomizing (U, T), into the ciphertext $U'' \leftarrow U^w$, $T'' \leftarrow (T/V)^w$ for a random $w \overset{\$}{\leftarrow} \mathbb{Z}_p^*$. It also provides a proof of validity: existence of $\mathsf{sk}\mathcal{H}$ and $w' = 1/w \bmod p$ such that

$$g^{\mathsf{sk}\mathcal{H}} = \mathsf{pk}\mathcal{H} \qquad U'^{-\mathsf{sk}\mathcal{H}} \cdot T''^{w'} = T/T' \qquad U''^{w'} = U \tag{5}$$

which guarantees the correct computation of (U'', T''), using the secret key associated to $\mathsf{pk}\mathcal{H}$.

Note that since (U, T) is an encryption of Z under the key h', which associated decryption key β is known to the user, and (U', T') is an encryption of $V = Z \cdot g^{\boldsymbol{f}\cdot\boldsymbol{m}-v}$ under the key $\mathsf{pk}\mathcal{H}$, (U'', T'') is an encryption of $(g^{\boldsymbol{f}\cdot\boldsymbol{m}-v})^w$ under the key h'.

Concretely, the helper sends (U'', T'') and 3 scalars for the proof (the challenge and the answers for the 2 witnesses $\mathsf{sk}\mathcal{H}$ and w').

Eventually, after having checked the proofs, the user can compute $K = T''/U''^{\beta}$. If $K = 1$, this means the answer is YES, otherwise the answer is NO. Indeed, as remarked above, (U'', T'') is an encryption of $(g^{\boldsymbol{f}\cdot\boldsymbol{m}-v})^w$ under the key $h' = g^\beta$. And so, $K = 1$ if $\boldsymbol{f} \cdot \boldsymbol{m} = v \bmod p$. Otherwise, $g^{\boldsymbol{f}\cdot\boldsymbol{m}-v} \neq 1_{\mathbb{G}}$, and so is a generator of $\mathbb{G}$. As a consequence, K is a random element in $\mathbb{G}$, since $w \neq 0 \bmod p$.

4.3 Decryption

Actually, we can also allow a regular decryption query $\mathsf{Decrypt}(\mathsf{pk}\mathcal{H}, C, \mathsf{sk}\boldsymbol{f})$: the above pair (B, A) is an ElGamal encryption of g^v, which can be decrypted with the help of $\mathcal{H}$, which knows the decryption key $\mathsf{sk}\mathcal{H}$. But again, we do not want to leak any information to $\mathcal{H}$.

Hence, the user does as above, and sends (W_0, U_1, T_1), (U_2, T_2), (h', U, T), and (U', T'), with the proof of correctness, but with $U' \leftarrow B \cdot g^{\rho'}$ and $T' \leftarrow A \cdot Z \cdot \mathsf{pk}\mathcal{H}^{\rho'}$ (without dividing by g^v). Then (U', T') is an encryption of $Z \cdot g^{\boldsymbol{f}\cdot\boldsymbol{m}}$, that $\mathcal{H}$ can decrypt (with a proof of correctness). Then, the user can remove Z to get $g^{\boldsymbol{f}\cdot\boldsymbol{m}}$, from which he can compute the discrete logarithm, since $C \in \Phi$ and $\boldsymbol{f} \in \Psi$.

We thus have again a verifiable decryption process, with the helper that remains *oblivious*. The same proofs as below for indistinguishability and query-privacy would indeed show that the helper does not learn anything about the plaintext of C, nor on $\boldsymbol{f}$, and the user does not learn anything else than $\boldsymbol{f} \cdot \boldsymbol{m}$ for the legitimate inputs.

4.4 Communication Complexity

During the TestDecrypt protocol, the user sends 10 group elements from $\mathbb{G}$ and $\ell + 9$ scalars from $\mathbb{Z}_p$, and the helper answers with 2 group elements from $\mathbb{G}$ and 3 scalars from $\mathbb{Z}_p$. During the Decrypt protocol, the user sends 10 group elements from $\mathbb{G}$ and $\ell + 9$ scalars from $\mathbb{Z}_p$, and the helper answers with 1 group elements from $\mathbb{G}$ and 2 scalars from $\mathbb{Z}_p$ (for the proof of valid ElGamal decryption). This assumes that the ciphertext C has been made publicly available, and that the user only needs to send its id number.

If the ciphertext C is not public, the user also needs to send it, raising the communication complexity during TestDecrypt or Decrypt by $2\ell + 2$ group elements. To certify the ciphertext being legitimate in the database, a signature by the owner of the database is also required, which also increases the communication load (but in this case, any basic signature scheme is acceptable).

4.5 Security Analysis

For the security proofs below, we do not consider the Decrypt-oracle, but just TestDecrypt-queries. However, similar security notions/proofs would hold.

QUERY PRIVACY. The only information sent by the user are C, which is thus explicit for the helper, ElGamal encryptions (commitments) of W_1 and g^v under g_τ, ElGamal encryption of Z, a (partial) randomized signature $W_0 \leftarrow \mathsf{sk}\boldsymbol{f}_0^\sigma$ and zero-knowledge proofs of validity. Under the DDH assumption in $\mathbb{G}$, the ciphertexts and randomized signature do not reveal any information on $\boldsymbol{f}$, v, and the outcome (which is equivalent to $Z = V$). Thanks to the zero-knowledge property of the proofs, they do not reveal anything either. □

DATA PRIVACY. The proof for this notion is quite similar to the adaptive security of the basic Function Encryption scheme from [ALS15,ALS16]. Hence, we just postpone it to the Appendix A for completeness.

INDISTINGUISHABILITY. In this proof, we have to show the existence of a simulator $\mathcal{S}$ that emulates $\mathcal{H}$ from the lists Φ, Ψ and the oracle Test, on the message M sent by the user, but without $\mathsf{sk}\mathcal{H}$. We will build this simulator with a sequence of games, starting from $\mathcal{H}$, knowing $\mathsf{sk}\mathcal{H}$, and show that we can do the same with Test, without knowing $\mathsf{sk}\mathcal{H}$. Since Test can only be asked on legitimate ciphertexts and functions, they are necessarily in $\mathcal{M}$ and $\mathcal{F}$, respectively (Encrypt and KeyDer indeed refuse to answer for values not in these spaces).

Game $\mathbf{G}_0$: This is the initial game, where the simulator knows the public keys and $\mathsf{sk}\mathcal{H}$. It thus mimics $\mathcal{H}$ for TestDecrypt-queries.

Game $\mathbf{G}_1$: In this game, the simulator generates $g_\tau = g^\tau$, which allows it to open the commitments (U_1, T_1) and (U_2, T_2) to get F_1 and F_2. The value F_1 should be the complement of W_0 for a valid signature of $\boldsymbol{f}$ and F_2 should be g^v. If (W_0, F_1) is not a valid signature for any vector $\boldsymbol{f} \in \Psi$, the simulator aborts.

We can show that under the $\mathtt{EUF-CMA}$ security of the signature scheme, the simulator never aborts: In case of abort in $\mathbf{G}_1$, this means that in $\mathbf{G}_0$, one TestDecrypt-query contains such an unrelated signature. Now, the simulator does not know anymore the signing key, but has access to the signing oracle to answer the KeyDer-queries. For such a TestDecrypt-query with unrelated signature, by rewinding the adversary, one can extract ρ, ρ', ρ_1, ρ_2, $\mathsf{sk}\boldsymbol{f}_1$, $\mathsf{sk}\boldsymbol{f}_2$, v, and $f_1, \ldots, f_\ell$ that satisfy all the relations, and namely $\boldsymbol{f}$ with relations (1):

$$g^{\rho_1} = U_1$$
$$\prod_i e(W_0, \tilde{Y}_1)^{f_i} \cdot e(g_\tau, \tilde{g})^{\rho_1} = e(T_1, \tilde{g}) / e(W_0, \tilde{X})$$

Since $g_\tau = g^\tau$ and F_1 is the plaintext encrypted in (U_1, T_1) under g_τ, we have

$$e(W_0, \tilde{X}) \cdot \prod_i e(W_0, \tilde{Y}_1)^{f_i} = e(T_1/U_1^\tau, \tilde{g}) = e(F_1, \tilde{g}).$$

Hence, (W_0, F_1) is a valid signature of a new vector $\boldsymbol{f}$, which breaks $\mathtt{EUF-CMA}$.

Game G$_2$: As a consequence, for any TestDecrypt-query, the commitments (U_1, T_1) and (U_2, T_2), allow to learn the unique vector $\boldsymbol{f}$ that satisfies the relations (1), unless one can break the $\mathtt{EUF-CMA}$ security of the signature scheme, by simply looking for it in the list Ψ, and a group element g^v (with v unknown).

Since only legitimate ciphertexts can be queried ($C \in \Phi$), we know that there exists a scalar r such that

$$C_0 = g_s^r \qquad C_0' = g_t^r \qquad C_i = \mathsf{pk}\mathcal{H}^{\alpha_i} g^{m_i} h_i^r \qquad D_i = g^{\alpha_i}.$$

Using relation (3), we know that $\prod_i h_i^{f_i} = g_s^{\mathsf{sk}\boldsymbol{f}_1} \cdot g_t^{\mathsf{sk}\boldsymbol{f}_2}$. While relation (4) guarantees that

$$\begin{aligned} U &= g^\rho \qquad U' = g^{\rho' + \boldsymbol{\alpha}\cdot\boldsymbol{f}} \\ T &= T' \cdot \mathsf{pk}\mathcal{H}^{-\rho'} \cdot h'^{\rho} \cdot \underbrace{C_0^{\mathsf{sk}\boldsymbol{f}_1} C_0'^{\,\mathsf{sk}\boldsymbol{f}_2} \cdot g^v \cdot \prod_i C_i^{-f_i}}_{=\prod_i h_i^{rf_i} \cdot g^v \cdot \mathsf{pk}\mathcal{H}^{-\boldsymbol{\alpha}\cdot\boldsymbol{f}} \cdot g^{-\boldsymbol{m}\cdot\boldsymbol{f}} \cdot \prod_i h_i^{-rf_i}} \\ &= T' \cdot h'^{\rho} \cdot \mathsf{pk}\mathcal{H}^{-\rho' - \boldsymbol{\alpha}\cdot\boldsymbol{f}} \cdot g^{v - \boldsymbol{m}\cdot\boldsymbol{f}}. \end{aligned}$$

If (U, T) encrypts some Z under h', then

$$T' = \mathsf{pk}\mathcal{H}^{\rho' + \boldsymbol{\alpha}\cdot\boldsymbol{f}} \cdot Z \cdot g^{\boldsymbol{m}\cdot\boldsymbol{f} - v} \qquad U = g^{\rho' + \boldsymbol{\alpha}\cdot\boldsymbol{f}}$$

As a consequence, (U', T') is an ElGamal encryption of $Z \cdot g^{\boldsymbol{m}\cdot\boldsymbol{f} - v}$, under $\mathsf{pk}\mathcal{H}$.

The simulator knows $\mathsf{sk}\mathcal{H}$, and so it can compute $V \leftarrow T'/U'^{\mathsf{sk}\mathcal{H}} = Z \cdot g^{\boldsymbol{m}\cdot\boldsymbol{f} - v}$. It can perfectly answer to the user, with V. Nevertheless, it will use the zero-knowledge property of the proofs to simulate them. More precisely, it honestly generates (U'', T''), but will simulate the proofs of validity.

Game G$_3$: In this final game, the simulator still knows τ such that $g_\tau = g^\tau$, but now uses the Test-oracle on legitimate ciphertexts and functions. It can thus extract $\boldsymbol{f}$ and g^v from (U_1, T_1) and (U_2, T_2). In order to ask the Test-oracle, it needs to compute the discrete logarithm v: if after time Time, it fails to compute v, the simulator considers that $v \neq \boldsymbol{f}\cdot\boldsymbol{m} \bmod p$ (we know that for any legitimate C and $\boldsymbol{f}$, the discrete logarithm of $g^{\boldsymbol{f}\cdot\boldsymbol{m}}$ can be computed in time less than Time, so we indeed have $v \neq \boldsymbol{f}\cdot\boldsymbol{m} \bmod p$ if time is larger). If it gets v, it asks the Test-oracle on $(C, \boldsymbol{f}, v)$, for $C \in \Phi$ and $\boldsymbol{f} \in \Psi$, and learns whether $\boldsymbol{f}\cdot\boldsymbol{m} = v \bmod p$ or not. If $\boldsymbol{f}\cdot\boldsymbol{m} = v \bmod p$, it generates $(U'' = g^w, T'' = h'^w)$, for $w \xleftarrow{\$} \mathbb{Z}_p$, and thus a random encryption of $1_{\mathbb{G}}$. Otherwise, it generates $(U'' = g^w, T'' = h'^{w'})$, for $w, w' \xleftarrow{\$} \mathbb{Z}_p$, and thus a random ciphertext. It also sends simulated proofs of validity. This is exactly as in the previous game. □

Verifiability. The verifiability property means that both the KeyDer-oracle and the TestDecrypt-oracle give verifiable answers: if they are not correct, the user can detect it, then it is equivalent to a Denial-of-Service attack.

The former oracle, on input $\boldsymbol{f} = (f_1, \ldots, f_\ell) \in \mathcal{F}$, samples $h \xleftarrow{\$} \mathbb{G}$, and outputs $\mathsf{sk}\boldsymbol{f} = (\mathsf{sk}\boldsymbol{f}_0 = h, \mathsf{sk}\boldsymbol{f}_1 = \boldsymbol{f}\cdot\boldsymbol{s} \bmod p, \mathsf{sk}\boldsymbol{f}_2 = \boldsymbol{f}\cdot\boldsymbol{t} \bmod p, \mathsf{sk}\boldsymbol{f}_3 = h^{x + \sum_i y_i f_i})$. As said above, one can check the validity:

$$\prod_i h_i^{f_i} = g_s^{\mathsf{sk}\boldsymbol{f}_1} \cdot g_t^{\mathsf{sk}\boldsymbol{f}_2} \qquad e(\mathsf{sk}\boldsymbol{f}_0, \tilde{X}\prod_i \tilde{Y}_i^{f_i}) = e(\mathsf{sk}\boldsymbol{f}_3, \tilde{g}).$$

The latter oracle, on input (W_0, U_1, T_1), (U_2, T_2), (h', U, T), (U', T'), as well as the proof of validity, also answers with (U'', T'') and the proof of validity. Hence, the user has the guarantee that the plaintext in (U'', T'') is $1_{\mathbb{G}}$ if and only if $\boldsymbol{f}\cdot\boldsymbol{m} = v \bmod p$. □

4.6 Test of Interval

The construction proposed and analyzed in the previous sections limits the user to ask queries one by one, which is both a waste of time and sometimes a too precise answer in case of positive one. In many applications, such as the decision for SVM in machine learning, one just wants to test whether the scalar product of the plaintext is in an interval $\{v, \ldots, v + L - 1\}$, without learning the exact value. We can extend it to handle this case, at a very low cost, by just revealing the length L of the interval to the helper.

Indeed, the user does exactly as before, until the computation of the pair (U', T'), that is an ElGamal encryption of $Z \cdot g^{\boldsymbol{m} \cdot \boldsymbol{f} - v}$ under $\mathsf{pk}\mathcal{H}$, and of the pair (U, T), that is an ElGamal encryption of Z under $h' = g^{\beta}$. He additionally sends L. The helper can easily derive all the $V_i = V \times g^{-i}$, for $i = 0, \ldots, L - 1$, where $V = T'/U'^{\mathsf{sk}\mathcal{H}}$. One can note that $V_i = Z \cdot g^{\boldsymbol{m} \cdot \boldsymbol{f} - v - i}$. It can also compute $U''_i \leftarrow U^{w_i}$ for a random $w_i \xleftarrow{\$} \mathbb{Z}_p$, and $T''_i \leftarrow (T/V_i)^{w_i}$. Then the pair (U''_i, T''_i) is a random ElGamal encryption of $g^{w_i(\boldsymbol{m} \cdot \boldsymbol{f} - v - i)}$.

When decrypting the ciphertexts $(U''_i, T''_i)_{i=0,\ldots,L-1}$, at most one encrypts $1_{\mathbb{G}}$, which means that $\boldsymbol{f} \cdot \boldsymbol{m} \in \{v, \ldots, v + L - 1\} \bmod p$. However, the index of the ciphertext that decrypts to $1_{\mathbb{G}}$ leaks $\boldsymbol{f} \cdot \boldsymbol{m}$, which is too much information. The server thus has to send the ciphertexts $(U''_i, T''_i)_{i=0,\ldots,L-1}$ in a permuted order, but also in verifiable way.

To this aim, it first computes all the (U'_i, T'_i) that correspond to encryption of V_i under $\mathsf{pk}\mathcal{H}$, as $U'_i = U'$ and $T'_i = T' \times g^{-i}$, for $i = 0, \ldots, L - 1$. Indeed, since (U', T') is an ElGamal encryption of $Z \cdot g^{\boldsymbol{m} \cdot \boldsymbol{f} - v}$ under $\mathsf{pk}\mathcal{H}$, (U'_i, T'_i) is an ElGamal encryption of $V_i = Z \cdot g^{\boldsymbol{m} \cdot \boldsymbol{f} - v - i}$ under $\mathsf{pk}\mathcal{H}$. And both the user and the server can compute these values. The server makes a shuffling, using the homomorphic property of the ElGamal encryption scheme and the associated proof of correctness, which can be done in sublinear size [BG12]. Then, it continues with each of these ciphertexts, to rerandomize (U, T) into (U''_i, T''_i) for $i = 0, \ldots, L - 1$. These ciphertexts are the same as above, but in a random order.

Then, among the L ciphertexts, at most one encrypts $1_{\mathbb{G}}$, which just reveals $\boldsymbol{f} \cdot \boldsymbol{m} \in \{v, \ldots, v + L - 1\} \bmod p$, but nothing else. This is at no additional cost for the user, and just linear in L for the helper, but still leaking no information excepted the considered ciphertext C and the length L of the interval: the larger L is, the less information is revealed, which is thus an important information for the helper to allow/restrict more queries.

Conclusion

In this work, we introduced a variant of functional encryption, that makes use of a third player, the *helper*, to make decryption interactive which allows fine tuning of the leakage of information, with a more restrictive policy. In addition, we allow decryption tests only, where one learns whether $\boldsymbol{f}(\boldsymbol{m}) \in V$, for a ciphertext C of $\boldsymbol{m}$, a function $\boldsymbol{f}$, and a set V, and not the decryption itself.

This paves the way for securing *machine learning* processes, that only require computations on aggregated data, but nowadays can only work if the data is accessible in clear. Specifically, our concrete scheme (Section 4) targets the inner-product class of functions, that is at the basis of the support vector machine class of algorithms, very famous in machine learning.

Hence, a concrete scenario would be that the owner of a sensitive database encrypts it using our scheme, then hands the encrypted database to its analytics contractor. This one can, with the help of the *oblivious helper* server located in the *cloud*, compute the necessary inner products (which list is anyway restricted by the owner of the database that gave some functional keys only) needed to train its algorithm, and then answer classifying queries. The privacy of each entry in the database remains protected, as the helper will prevent any attempt at asking too many queries on a single ciphertext. And the queries remain oblivious to the helper.

Several improvements could be made to increase efficiency and/or privacy. Notably, the helper could be distributed among several players, while maintaining verifiability. This can easily be done using Shamir secret sharing [Sha79] on pure decryption queries, but it proves tricky for decryption test, as our construction relies strongly on the random power the helper adds to limit the leakage of information, which cannot be easily synchronized between several helpers.

Acknowledgments

This work was supported in part by the European Research Council (ERC) under the European Community's Seventh Framework Programme (FP7/2007-2013 Grant Agreement no. 339563 – CryptoCloud).

References

ABDP15. M. Abdalla, F. Bourse, A. De Caro, and D. Pointcheval. Simple functional encryption schemes for inner products. In *PKC 2015*, *LNCS* 9020, pages 733–751. Springer, Heidelberg, March / April 2015.

ABDP16. M. Abdalla, F. Bourse, A. De Caro, and D. Pointcheval. Better security for functional encryption for inner product evaluations. Cryptology ePrint Archive, Report 2016/011, 2016. http://eprint.iacr.org/2016/011.

ALS15. S. Agrawal, B. Libert, and D. Stehle. Fully secure functional encryption for inner products, from standard assumptions. Cryptology ePrint Archive, Report 2015/608, 2015. http://eprint.iacr.org/2015/608.

ALS16. S. Agrawal, B. Libert, and D. Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In *CRYPTO 2016, Part III*, *LNCS* 9816, pages 333–362. Springer, Heidelberg, August 2016.

BCP14. E. Boyle, K.-M. Chung, and R. Pass. On extractability obfuscation. In *TCC 2014*, *LNCS* 8349, pages 52–73. Springer, Heidelberg, February 2014.

BDPR98. M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *CRYPTO'98*, *LNCS* 1462, pages 26–45. Springer, Heidelberg, August 1998.

BG12. S. Bayer and J. Groth. Efficient zero-knowledge argument for correctness of a shuffle. In *EUROCRYPT 2012*, *LNCS* 7237, pages 263–280. Springer, Heidelberg, April 2012.

BJK15. A. Bishop, A. Jain, and L. Kowalczyk. Function-hiding inner product encryption. In *ASIACRYPT 2015, Part I*, *LNCS* 9452, pages 470–491. Springer, Heidelberg, November / December 2015.

BR93. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

BSW11. D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In *TCC 2011*, *LNCS* 6597, pages 253–273. Springer, Heidelberg, March 2011.

CL04. J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO 2004*, *LNCS* 3152, pages 56–72. Springer, Heidelberg, August 2004.

CS97. J. Camenisch and M. Stadler. Efficient group signature schemes for large groups (extended abstract). In *CRYPTO'97*, *LNCS* 1294, pages 410–424. Springer, Heidelberg, August 1997.

ElG84. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO'84*, *LNCS* 196, pages 10–18. Springer, Heidelberg, August 1984.

GGH$^+$13. S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

GGHZ16. S. Garg, C. Gentry, S. Halevi, and M. Zhandry. Functional encryption without obfuscation. In *TCC 2016-A, Part II*, *LNCS* 9563, pages 480–511. Springer, Heidelberg, January 2016.

GPS08. S. D. Galbraith, K. G. Paterson, and N. P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

HDO$^+$98. M. A. Hearst, S. T. Dumais, E. Osman, J. Platt, and B. Scholkopf. Support vector machines. *IEEE Intelligent Systems and their Applications*, 13(4):18–28, 1998.

JDM00. A. K. Jain, R. P. W. Duin, and J. Mao. Statistical pattern recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):4–37, 2000.

LRSW99. A. Lysyanskaya, R. L. Rivest, A. Sahai, and S. Wolf. Pseudonym systems. In *SAC 1999*, *LNCS* 1758, pages 184–199. Springer, Heidelberg, August 1999.

O'N10. A. O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010. http://eprint.iacr.org/2010/556.

PS96. D. Pointcheval and J. Stern. Security proofs for signature schemes. In *EUROCRYPT'96*, *LNCS* 1070, pages 387–398. Springer, Heidelberg, May 1996.

PS00. D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000.

PS16. D. Pointcheval and O. Sanders. Short randomizable signatures. In *CT-RSA 2016*, *LNCS* 9610, pages 111–126. Springer, Heidelberg, February / March 2016.

Sch90a. C.-P. Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO'89*, *LNCS* 435, pages 239–252. Springer, Heidelberg, August 1990.

Sch90b. C.-P. Schnorr. Efficient identification and signatures for smart cards (abstract) (rump session). In *EUROCRYPT'89*, *LNCS* 434, pages 688–689. Springer, Heidelberg, April 1990.

Sch91. C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

Sha79. A. Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

SS02. B. Schölkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond.* MIT press, 2002.

SW05. A. Sahai and B. R. Waters. Fuzzy identity-based encryption. In *EUROCRYPT 2005*, *LNCS* 3494, pages 457–473. Springer, Heidelberg, May 2005.

Wat15. B. Waters. A punctured programming approach to adaptively secure functional encryption. In *CRYPTO 2015, Part II*, *LNCS* 9216, pages 678–697. Springer, Heidelberg, August 2015.

A Proof of Data Privacy

The proof is inspired from [ALS15, ALS16]. Since $\mathsf{sk}\mathcal{H}$ is known to the adversary:

Game $\mathbf{G}_0$: This is the initial game, where the adversary, and thus the simulator, knows all the secret keys and random coins. The simulator encrypts $\boldsymbol{m} = \boldsymbol{m}^b$ for a random bit $b \overset{\$}{\leftarrow} \{0,1\}$ (the same in all this experiment) and random $r \overset{\$}{\leftarrow} \mathbb{Z}_p$, $\boldsymbol{\alpha} \overset{\$}{\leftarrow} \mathbb{Z}_p^\ell$:

$$C_0 = g_s^r, \quad C_0' = g_t^r, \quad C_i = \mathsf{pk}\mathcal{H}^{\alpha_i} \cdot g^{m_i} \cdot h_i^r, \quad D_i = g^{\alpha_i}$$

and the adversary outputs its guess b' for b:

$$\mathsf{Adv}^{\mathsf{ind}}(\mathcal{A}) = \mathsf{Adv}^{\mathsf{ind}}_{\mathbf{G}_0}(\mathcal{A}).$$

Game $\mathbf{G}_1$: The simulator modifies the generation of the challenge ciphertext, with a Diffie-Hellman pair $(A = g_s^r, B = g_t^r)$:

$$\begin{aligned} C_0 &= A, & C_0' &= B, \\ C_i &= \mathsf{pk}\mathcal{H}^{\alpha_i} \cdot g^{m_i} \cdot A^{s_i} \cdot B^{t_i}, & D_i &= g^{\alpha_i}. \end{aligned}$$

This does not change C_i since $h_i = g_s^{s_i} g_t^{t_i}$:

$$\mathsf{Adv}^{\mathsf{ind}}_{\mathbf{G}_0}(\mathcal{A}) = \mathsf{Adv}^{\mathsf{ind}}_{\mathbf{G}_1}(\mathcal{A}).$$

Game $\mathbf{G}_2$: The simulator does the same, but with a random pair $(A, B) \overset{\$}{\leftarrow} \mathbb{G}^2$:

$$\begin{aligned} C_0 &= A, & C_0' &= B, \\ C_i &= \mathsf{pk}\mathcal{H}^{\alpha_i} \cdot g^{m_i} \cdot A^{s_i} \cdot B^{t_i}, & D_i &= g^{\alpha_i}. \end{aligned}$$

Under the DDH in $\mathbb{G}$, this change cannot be detected by the adversary. Hence:

$$\mathsf{Adv}^{\mathsf{ind}}_{\mathbf{G}_1}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{ind}}_{\mathbf{G}_2}(\mathcal{A}) + \mathsf{Adv}^{\mathsf{DDH}}(\mathcal{A}).$$

We will now specifically analyze $\mathbf{G}_2$, and show that the adversary has no information at all about b in this game: even a more powerful adversary, $\mathcal{A}^*$, able to compute discrete logarithms (and therefore knowing $\boldsymbol{\alpha}$), cannot learn anything about b.
Indeed, the only information the adversary gets on the tuple $(b, \boldsymbol{s}, \boldsymbol{t})$ is:
– from the public key $(h_i)_i$: $\boldsymbol{a} = \boldsymbol{s} + z\boldsymbol{t}$, where $g_t = g_s^z$ and $h_i = g_s^{a_i}$;
– from the functional keys $\mathsf{sk}\boldsymbol{f}$: $b_{\boldsymbol{f}} = \boldsymbol{f} \cdot \boldsymbol{s} \bmod p$ and $b'_{\boldsymbol{f}} = \boldsymbol{f} \cdot \boldsymbol{t} \bmod p$.

From $C_i/\mathsf{pk}\mathcal{H}^{\alpha_i} = g_s^{c_i}$, it also knows $\boldsymbol{c} = u\boldsymbol{m} + r\boldsymbol{s} + z'r\boldsymbol{t} \bmod p$, where $g = g_s^u$, $A = g_s^r$ and $B = A^{z'}$, with $z' \neq z \bmod p$ (with overwhelming probability, $B \neq A^z$).
However, the restriction on the key derivation queries is that for each $\boldsymbol{f}$, $\boldsymbol{f} \cdot \boldsymbol{m}_0 = \boldsymbol{f} \cdot \boldsymbol{m}_1$: if we denote $\boldsymbol{\mu} = \boldsymbol{m}_b - \boldsymbol{m}_{1-b}$ in $\mathbb{Z}_p^\ell$, we know that $\boldsymbol{f} \cdot \boldsymbol{\mu} = 0 \bmod p$. We can now define $\boldsymbol{t}' = \boldsymbol{t} + u/r(z' - z) \times \boldsymbol{\mu}$. With exactly, the same probability, the simulator could have used $\boldsymbol{t}'$ and $\boldsymbol{s}' = \boldsymbol{a} - z\boldsymbol{t}' = \boldsymbol{s} - uz/r(z' - z) \times \boldsymbol{\mu}$: the public key would have been the same and the functional keys too (since they have been provided for functions $\boldsymbol{f}$ such that $\boldsymbol{f} \cdot \boldsymbol{\mu} = 0 \bmod p$), and the challenge ciphertext could then be seen as with $\boldsymbol{c} = u\boldsymbol{m} + r\boldsymbol{s}' + uz/(z'-z) \times \boldsymbol{\mu} + z'r\boldsymbol{t}' - uz'/(z'-z) \times \boldsymbol{\mu} = u(\boldsymbol{m} - \boldsymbol{\mu}) + r\boldsymbol{s}' + z'r\boldsymbol{t}'$. It corresponds to a ciphertext of $\boldsymbol{m}_b - \boldsymbol{\mu} = \boldsymbol{m}_b - (\boldsymbol{m}_b - \boldsymbol{m}_{1-b}) = \boldsymbol{m}_{1-b}$. This shows that the view of the adversary would have been exactly the same with $(1 - b, \boldsymbol{s}', \boldsymbol{t}')$, as with $(b, \boldsymbol{s}, \boldsymbol{t})$. And so, it does not leak any information about b. Which means that:

$$\mathsf{Adv}^{\mathtt{ind}}_{\mathbf{G}_2}(\mathcal{A}) \leq \mathsf{Adv}^{\mathtt{ind}}_{\mathbf{G}_2}(\mathcal{A}^*) = 0$$

Eventually, the best advantage an adversary can get against the data privacy security notion within time t is bounded by the advantage one can get within time t against the DDH problem in $\mathbb{G}$.

Hence:

$$\mathsf{Adv}^{\mathtt{ind}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{DDH}}(\mathcal{A}) \qquad \square$$