Miguel Ambrona, Dario Fiore, and Claudio Soriente

Controlled Functional Encryption Revisited: Multi-Authority Extensions and Efficient Schemes for Quadratic Functions

Abstract: In a Functional Encryption scheme (FE), a trusted authority enables designated parties to compute specific functions over encrypted data. As such, FE promises to break the tension between industrial interest in the potential of data mining and user concerns around the use of private data. FE allows the authority to decide *who* can compute and *what* can be computed, but it does not allow the authority to control *which* ciphertexts can be mined. This issue was recently addressed by Naveed et al., that introduced so-called *Controlled* Functional encryption (or C-FE), a cryptographic framework that extends FE and allows the authority to exert fine-grained control on the ciphertexts being mined. In this work we extend C-FE in several directions. First, we distribute the role of (and the trust in) the authority across several parties by defining multi-authority C-FE (or mC-FE). Next, we provide an efficient instantiation that enables computation of quadratic functions on inputs provided by multiple data-owners, whereas previous work only provides an instantiation for linear functions over data supplied by a single data-owner and resorts to garbled circuits for more complex functions. Our scheme leverages CCA2 encryption and linearly-homomorphic encryption. We also implement a prototype and use it to showcase the potential of our instantiation.

Keywords: controlled function encryption, computation over encrypted data

DOI 10.2478/popets-2021-0003
Received 2020-05-31; revised 2020-09-15; accepted 2020-09-16.

Miguel Ambrona: NTT Secure Platform Laboratories, E-mail: miguel.ambrona.fu@hco.ntt.co.jp
Dario Fiore: IMDEA Software Institute, E-mail: dario.fiore@imdea.org
Claudio Soriente: NEC Laboratories Europe, E-mail: claudio.soriente@nec.eu

1 Introduction

Functional Encryption (FE) [14, 40] allows authorized parties to compute a function over encrypted data and learn nothing beyond the function output. In a typical FE scenario, an authority distributes secret function keys; an entity holding a function key for function f can apply it over a ciphertext (encrypting data m), to learn $f(m)$ and nothing else. The blueprint of FE fits the description of many data mining applications, where data utility must be balanced with data privacy. For example, users may encrypt their DNA material and send it to a clinic to check for specific genetic markers (i.e., the function). An authority, say the national health system, may grant to that clinic the function key to carry out the computation. As long as the authority is trusted, the clinic may only compute the function(s) for which it received function key(s).

As such, the authority regulates *who* can compute and *what* can be computed over encrypted data. Yet, FE does not allow the authority to control on *which* ciphertexts a function key can be used. For example, the clinic may hold the function key for kinship tests over encrypted DNAs and Alice may want her DNA to be tested only versus Bob's. Such an access control may require either increasing the complexity of the function to be computed or introducing mechanisms beyond FE.

This shortcoming of FE was the focus of *Controlled Functional Encryption* (C-FE), introduced by Naveed et al. [37]. This is a special flavor of FE where the decryption algorithm (i.e., the one that computes $f(m)$) becomes a two-party protocol between the party holding the function key and the authority, where only the former learns the output of the computation. That way, the authority can exert fine-grained access control on the ciphertexts fed as input to the decryption procedure. For example, Alice can attach a policy to her encrypted DNA specifying that it may only be used to test kinship versus Bob's. The authority can, therefore, deny any computation that involves Alice's DNA and those belonging to individuals others than Bob.

The authors of [37] provide a simulation-based definition of C-FE that models malicious *clients* (i.e., parties computing functions over encrypted data) and a single, honest-but-curious (hbc) authority. They propose a lightweight C-FE for linear functions that leverages a CCA2 encryption scheme. They also propose a C-FE for arbitrary functions that leverages garbled circuits. Both instantiations are "single-input", i.e., they can accommodate encrypted data from a single data producer. As such, the C-FE instantiations of [37] cannot be used in the kinship test example, unless Bob's DNA is available in the clear to the client and the authority.

Our Contribution. In this paper, we continue the study of C-FE and extend it in several directions.

First, we provide a novel simulation-based definition that accounts for multiple authorities and ensures that a client can obtain no information from a ciphertext, unless all the authorities allow so. We argue that a *multi-authority* C-FE (or mC-FE) dramatically increases the security of its single-authority counterpart where the one authority unilaterally enforces access control to data and represents a single point of failure. The multi-authority model turns useful in applications where decisions on the generation of a function key or on its use over specific ciphertexts need to be taken by distinct entities, or in cases where one wants to simply and efficiently split the trust on the authority's secret-key material across different parties.

Second, we propose a mC-FE scheme for *multi-input* quadratic functions. Multi-input functions are necessary to implement application scenarios like the kinship test described earlier where functions are computed over data produced by multiple independent parties (e.g., Alice and Bob). Quadratic functions enable to compute several *statistic functions* such as average, variance, covariance or simple hypothesis testings; they could also be used for computing distances and areas[1] between points on a map (e.g., for location-based applications).

We showcase our mC-FE instantiations by implementing a prototype and evaluating its performance (Section 6) to, for example, classify encrypted images of handwritten digits of the MNIST dataset [33] or find correlations between age and blood pressure [44]. Our results show that the described C-FE constructions are efficient and can be used for real-world applications.

Our Techniques. In order to design our mC-FE for quadratic functions we start from a technique by Catalano and Fiore [18] for compiling a linearly-homomorphic encryption into an homomorphic encryption for quadratic functions. In its essence, this technique shows how to transform an encryption of m, of the form $(m-r,\ \mathsf{Enc}(r))$ for a random r, into $\mathsf{Enc}(f(m)-f(r))$, where f is a quadratic function and Enc a linearly-homomorphic encryption. To begin with, let us show how this can be used in a single-authority setting. The basic idea[2] is that the data-owner gives r to the client and $(m-r,\ \mathsf{Enc}(r))$ to the authority; if the authority grants permission for computing a function f, it computes homomorphically $\mathsf{Enc}(f(m)-f(r))$ and gives it to the client; the client can then decrypt it, and by computing $f(r)$ can recover the result $f(m)$.

Extending this idea to work with multiple authorities presents a number of challenges stemming from the fact that in the multi-authority setting we need to ensure security even if the adversary controls the client and (up to) all but one authority. Let us discuss why some attempts do not work.

A first attempt may be to let each authority receive $(m-r,\ \mathsf{Enc}(r))$ and compute $\mathsf{Enc}(f(m)-f(r))$; then they may altogether perform a threshold decryption on this ciphertext in order to provide the client with $f(m)-f(r)$. However, now the client would need r to recover the final result, which is a no-go for security since in the multi-authority setting the client may collude with any of the authorities and thus recover the input m from $m-r$.

A second attempt may consider a solution similar to the previous one in which we remove the need for r and use directly an homomorphic encryption for quadratic functions. So, each authority receives $\mathsf{Enc}(m)$, computes homomorphically $\mathsf{Enc}(f(m))$ and then all authorities and the client run a threshold decryption on this ciphertext. The problem with this solution is that threshold decryption requires a common input ciphertext, namely every party must produce the same $\mathsf{Enc}(f(m))$. This is possible only if the homomorphic encryption is deterministic, which is also not an option because without a proper re-randomization $\mathsf{Enc}(f(m))$ may leak information about the original input m. Clearly, one could also achieve this process by having all authorities and the client run a general-purpose MPC protocol, but we are not aware of any such protocol that is efficient and requires minimal, asynchronous, interaction.

In our work we propose a solution that proceeds in a way similar to the first attempt and solves its technical challenges by using a form of secret sharing of

1 For example, by using Gauss' shoelace formula.

2 For simplicity of presentation, here we omit several details.

the message across all authorities and a combination of threshold decryption with secret-sharing reconstruction. Notably, our mC-FE for quadratic function can be instantiated using a variant of the ElGamal scheme. This is the main technical novelty of our solution; we refer to Section 5.2 for more details.

2 Related Work

Functional Encryption. The notion of Functional Encryption was put forth in [14, 40] where general syntax and both IND-based and SIM-based security definitions were provided. The first ensures that encryptions of any two messages x_0, x_1 are indistinguishable for an adversary that only has access to functional keys for functions f that satisfy $f(x_0) = f(x_1)$. On the other hand, simulation-based security guarantees the existence of a simulator that, given a polynomial number of function evaluations over data x, can simulate the view of an adversary that has an encryption of x and the corresponding functional keys.

Schemes for general functionalities [16, 28, 29, 45] usually rely on strong building blocks such as indistinguishability obfuscation (iO) or multilinear maps. In particular, given recent results [34] there is strong evidence that iO is necessary for FE for just cubic functions. Custom schemes for linear [3, 6, 11, 24] and quadratic [9] functions can rely on simpler assumptions.

An FE scheme whose encryption algorithm uses secret information is said to be secret-key [4, 5], otherwise it is known as public-key FE [3]. Function-private FE (functional keys do not leak information on the function being computed) have been studied in both the private-key setting [17, 43] and in the public-key one [12, 13].

Some works explore the notion of multi-input FE and multi-client FE (where functions are computed over ordered sets of ciphertexts) [30]. Alternative constructions in this setting have been proposed for both general functionalities [8, 30] and restricted classes of functions [2, 4, 5, 20]. We note that in some multi-client FE schemes [20], the encryption routine takes an additional *label* as input so that, given a function key, the corresponding function can be computed only on ciphertexts that were generated with the same label. As such, multi-client FE provides basic access control to the ciphertexts that can be mined with a given function key—one of the goals of C-FE. Nevertheless, complex access policies beyond string matching are not possible in multi-client FE, whereas C-FE allows for access policies of arbitrary complexity. Further, in multi-client FE schemes [2, 20] encryption is secret key and data-producers are required to act also as authorities, i.e., it is not possible to release encrypted data and go (permanently) offline, whereas in C-FE, encryption is public-key and data-owners are not required to be online after encrypting their data.

Other works try to reduce the trust on a single authority, e.g., Chandran et al. [19], define and build multi-authority FE for arbitrary functionalities (based on iO). Previous work has also tried to obtain efficient FE by combining cryptography with trusted hardware [27].

Mining encrypted data. A number of works allow to compute statistics on encrypted data. For example [35] allows a server to compute statistics on aggregated data supplied by a large population. However, their system requires coordination among all data producers. PRIO [21] also allows computation of statistic over private data but uses multiple servers (each holding a share of the private data) to carry out the computation. In the context of recommender systems, the authors of [39] propose a scheme (based on garbled circuits) where a "crypto-service provider" allows a recommender system to compute matrix factorization of private inputs supplied as ciphertexts by the users.

Classification over encrypted data is an emerging topic [15, 31, 36]. However, in all of these works the data suppliers are those who perform the classification over their own (encrypted) data. Differently, C-FE allows a third party (independent of the data supplier) to perform the computation, a functionality that is more difficult to achieve.

3 Preliminaries

3.1 Notation

For an integer $n \in \mathbb{N}$, we denote by $[n]$ the range of integers $\{1, \ldots, n\}$. We say that a function with range in $[0, 1]$ is *negligible* if it is asymptotically smaller than the inverse of any polynomial. We say that a function δ with range in $[0, 1]$ is *overwhelming* if $1 - \delta$ is negligible. We write $y \leftarrow A(x)$ to denote that y is the output of algorithm A on input x. For a vector $\boldsymbol{x}$ of size ℓ, we abuse notation and write $\boldsymbol{y} \leftarrow A(\boldsymbol{x})$, to denote $y_1 \leftarrow A(x_1); \ldots; y_\ell \leftarrow A(x_\ell)$. If A takes two arguments, we sometimes write $A_{x_1}(x_2)$ to denote $A(x_1, x_2)$. For algorithm A and O, we write $A^O(\cdot)$ to denote the fact that A has oracle access to function O.

3.2 Public-Key Encryption

We define public-key encryption (PKE) and the standard security notions that we use in our security proofs.

Definition 1 (Public-Key Encryption)**.** *A* public-key encryption *scheme is a tuple of three efficiently computable algorithms:*

- $\mathsf{KeyGen}(1^\kappa)$*: on input the security parameter, outputs a key pair* $(\mathsf{pk}, \mathsf{sk})$*. The public key,* pk*, includes a description of the message space* $\mathcal{M}$ *and the ciphertext space* $\mathcal{C}$*.*
- $\mathsf{Enc}(\mathsf{pk}, m)$*: on input the public key and a message* $m \in \mathcal{M}$*, outputs a ciphertext* ct*.*
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$*: given the secret key and a ciphertext, outputs a message* m*.*

A PKE scheme is said *correct* if there exists negligible function ϵ in κ such that for every (sufficiently large $\kappa \in \mathbb{N}$), for $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa)$ and for every $m \in \mathcal{M}$:

$$\Pr\left[\mathsf{Dec}(\mathsf{sk},\ \mathsf{Enc}(\mathsf{pk}, m)) = m\right] \geq 1 - \epsilon(\kappa)$$

where the probability is taken over the coins of KeyGen and Enc.

Definition 2 (Security of encryption)**.** *We say that a public-key encryption scheme* $\mathcal{E}$ *is* xxx secure*, for* $\mathsf{xxx} \in \{\mathsf{IND\text{-}CPA}, \mathsf{IND\text{-}CCA1}, \mathsf{IND\text{-}CCA2}\}$*, if there exists a negligible function* ϵ *in* κ *such that for every probabilistic polynomial-time (p.p.t.) stateful adversary* $\mathcal{A}$*, and for every (sufficiently large)* $\kappa \in \mathbb{N}$*,* $\mathsf{Adv}^{\mathsf{xxx}}_{\mathcal{E},\mathcal{A}}(\kappa) \leq \epsilon(\kappa)$*, where*

$$\mathsf{Adv}^{\mathsf{xxx}}_{\mathcal{E},\mathcal{A}}(\kappa) \coloneqq \left|\Pr[\mathsf{Exp}^{\mathsf{xxx}}_{\mathcal{E},\mathcal{A}}(\kappa) = 1] - \tfrac{1}{2}\right|$$

and the experiment $\mathsf{Exp}^{\mathsf{xxx}}_{\mathcal{E},\mathcal{A}}(\kappa)$ *is defined as:*

$$\mathsf{Exp}^{\mathsf{xxx}}_{\mathcal{E},\mathcal{A}}(\kappa) \coloneqq \left\{\begin{array}{r} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa) \\ (m_0, m_1) \leftarrow \mathcal{A}^{\mathcal{O}_1}(\mathsf{pk}) \\ b \xleftarrow{\$} \{0,1\} \\ \mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b) \\ b' \leftarrow \mathcal{A}^{\mathcal{O}_2}(\mathsf{ct}^*) \\ \text{if } b = b' \wedge |m_0| = |m_1| \text{ output } 1, \\ \text{otherwise output } 0. \end{array}\right.$$

where $\mathcal{O}_1$*,* $\mathcal{O}_2$ *are decryption oracles that, on input a ciphertext* ct*, return* $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$*. Oracle* $\mathcal{O}_2$ *returns* $\perp$ *if it is called on the target ciphertext* ct^**. In the* IND-CCA2 *experiment, both oracles are present. In the case of* IND-CCA1*, only* $\mathcal{O}_1$ *is available, while in the* IND-CPA *experiment, neither* $\mathcal{O}_1$ *nor* $\mathcal{O}_2$ *are available to* $\mathcal{A}$*.*

3.3 Homomorphic Encryption

An homomorphic encryption scheme is an encryption scheme equipped with a procedure to perform a computation on ciphertexts, producing a ciphertext for a value that equals the result of the computation if it were performed on the plaintexts.

In this work we use *linearly-homomorphic encryption* (LHE) as a building block, an homomorphic scheme where the operations allowed are: *addition* of ciphertexts, *addition of constants* to ciphertexts and *multiplication of constants* by ciphertexts. As in other works [10, 18], we consider schemes where messages are in certain ring $(\mathcal{M}, +, \cdot)$.

Definition 3 (Linearly-homomorphic encryption)**.** *A* linearly-homomorphic encryption *scheme is a tuple of six efficiently computable algorithms:*

- $\mathsf{KeyGen}(1^\kappa)$*: on input the security parameter, outputs a key pair* $(\mathsf{pk}, \mathsf{sk})$*. The public key,* pk*, includes a description of the message space* $\mathcal{M}$*.*
- $\mathsf{Enc}(\mathsf{pk}, m)$*: on input the public key and a message* $m \in \mathcal{M}$*, outputs a ciphertext* ct*.*
- $\mathsf{Eval.add}(\mathsf{pk}, \mathsf{ct}_1, \mathsf{ct}_2)$*: on input the public key, and two ciphertexts* ct_1*,* ct_2*, outputs a ciphertext.*
- $\mathsf{Eval.add\text{-}const}(\mathsf{pk}, m, \mathsf{ct})$*: on input the public key, a plaintext* m *and a ciphertext* ct*, outputs a ciphertext.*
- $\mathsf{Eval.mul\text{-}const}(\mathsf{pk}, m, \mathsf{ct})$*: on input the public key, a plaintext* m *and a ciphertext* ct*, outputs a ciphertext.*
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$*: given the secret key and a ciphertext, outputs a message* m *or* $\perp$*.*

A linearly-homomorphic encryption scheme is *correct* if for all $m, \widehat{m} \in \mathcal{M}$, $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$, $\widehat{\mathsf{ct}} \leftarrow \mathsf{Enc}(\mathsf{pk}, \widehat{m})$, the following probabilities are overwhelming in κ:

$$\begin{array}{l} \Pr\left[\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) = m\right] \\ \Pr\left[\mathsf{Dec}(\mathsf{sk}, \mathsf{Eval.add}(\mathsf{pk}, \mathsf{ct}, \widehat{\mathsf{ct}})) = m + \widehat{m}\right] \\ \Pr\left[\mathsf{Dec}(\mathsf{sk}, \mathsf{Eval.add\text{-}const}(\mathsf{pk}, m, \widehat{\mathsf{ct}})) = m + \widehat{m}\right] \\ \Pr\left[\mathsf{Dec}(\mathsf{sk}, \mathsf{Eval.mul\text{-}const}(\mathsf{pk}, m, \widehat{\mathsf{ct}})) = m \cdot \widehat{m}\right] \end{array}$$

for every honestly generated pair of keys $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa)$. (The probabilities are taken over the coins of all probabilistic algorithms involved.)

For arbitrary linear functions $f : \mathcal{M}^\ell \to \mathcal{M}$ of arity ℓ, we can define (from $\mathsf{Eval.add}$, $\mathsf{Eval.add\text{-}const}$, $\mathsf{Eval.mul\text{-}const}$) a procedure $\mathsf{Eval}(f, \mathsf{pk}, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)$ that, on input f, a public key and ℓ ciphertexts produces a

ciphertext ct that decrypts to $f(m_1,\ldots,m_\ell)$, where m_i is the plaintext encrypted in ct_i, for $i \in [\ell]$.

For *security* we require *semantic security*, equivalent to IND-CPA (Definition 2) and *circuit privacy*. Roughly, the former guarantees that ciphertexts leak no information about the plaintexts they were created from, while the latter guarantees that the Eval procedure does not leak information about the original plaintexts encoded in the processed ciphertexts.

Definition 4 (Circuit privacy)**.** *A linearly-homomorphic encryption scheme $\mathcal{H}$ satisfies* circuit privacy *if there exists a p.p.t. simulator $\mathcal{H}.\mathsf{Sim}$ such that for any pair of keys $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa)$, any linear circuit $f : \mathcal{M}^\ell \to \mathcal{M}$, all messages $m_1,\ldots,m_\ell \in \mathcal{M}$ and ciphertexts $\mathsf{ct}_1,\ldots,\mathsf{ct}_\ell$ built as $\mathsf{ct}_i \coloneqq \mathsf{Enc}(\mathsf{pk},m_i)$, $\forall i \in [\ell]$:*

$$\mathcal{H}.\mathsf{Eval}(f,\mathsf{pk},\mathsf{ct}_1,\ldots,\mathsf{ct}_\ell) \equiv \mathcal{H}.\mathsf{Sim}(1^\kappa,\mathsf{pk},f(m_1,\ldots,m_\ell))$$

where $\equiv$ denotes equality of distributions.

Remark. For general homomorphic encryption schemes, it is common to relax the above definition and admit statistically close distributions, however, it is not hard to achieve perfect circuit privacy for LHE schemes.

Extra notation. For two ciphertexts $\mathsf{ct},\widehat{\mathsf{ct}}$, we denote by $\mathsf{ct} \boxplus_{\mathsf{pk}} \widehat{\mathsf{ct}}$ the result of $\mathsf{Eval.add}(\mathsf{pk},\mathsf{ct},\widehat{\mathsf{ct}})$. Analogously, given a message $m \in \mathcal{M}$ and a ciphertext ct, we denote by $m \oplus_{\mathsf{pk}} \mathsf{ct}$ and $m \otimes_{\mathsf{pk}} \mathsf{ct}$ the result of $\mathsf{Eval.add\text{-}const}(\mathsf{pk},m,\mathsf{ct})$ and $\mathsf{Eval.mul\text{-}const}(\mathsf{pk},m,\mathsf{ct})$ respectively. When the public key is clear form the context, we simply write $\mathsf{ct} \boxplus \widehat{\mathsf{ct}}$, $m \oplus \mathsf{ct}$ or $m \otimes \mathsf{ct}$. Note that all these homomorphic operations may be probabilistic to allow circuit privacy.

3.4 Degree-2 Homomorphic Encryption from LHE

In [18], Catalano and Fiore proposed a method for compiling a linearly-homomorphic encryption scheme into an homomorphic encryption scheme that supports degree-2 computations. Although the technique of [18] did not yield fully compact ciphertexts (i.e., ciphertexts of fixed size, independently of the number of inputs in the computation), Barbosa et al. [10] later extended it to achieve compact ciphertexts in a different model called labeled homomorphic encryption. Our constructions build on the technique from [10], without however using the model of labeled homomorphic encryption.

The techniques from [10, 18] require that the message space be what they define as *public-space*, which can be informally described as a publicly known finite and commutative ring where it is possible to efficiently sample elements uniformly at random. The authors consider ciphertexts of the form $\mathsf{ct} = (m - r, \mathsf{Enc}(r))$, where $m \in \mathcal{M}$, r is chosen uniformly at random in $\mathcal{M}$ and "$-$" denotes the inverse operation of the addition, "$+$", in the ring. They observe that, for such ciphertexts, it is possible to perform degree-2 homomorphic computations. More precisely, given two ciphertexts $\mathsf{ct}_1 = (m_1 - r_1, \mathsf{Enc}(r_1))$ and $\mathsf{ct}_2 = (m_2 - r_2, \mathsf{Enc}(r_2))$ and given a degree-2 polynomial $f(x,y) \in \mathcal{M}[X,Y]$, it is possible to efficiently compute $\mathsf{Enc}(f(m_1,m_2) - f(r_1,r_2))$. For example, let $f(x,y) = axy + bx + cy + d$, for $a,b,c,d \in \mathcal{M}$ (d could actually be ignored):

1. Compute $\alpha \coloneqq (m_1 - r_1) \cdot (m_2 - r_2)$.
2. Set $\beta_1 \coloneqq (m_1 - r_1) \otimes \mathsf{Enc}(r_2) = \mathsf{Enc}(m_1 r_2 - r_1 r_2)$.
3. Set $\beta_2 \coloneqq (m_2 - r_2) \otimes \mathsf{Enc}(r_1) = \mathsf{Enc}(r_1 m_2 - r_1 r_2)$.
4. Combine $\gamma \coloneqq \alpha \oplus \beta_1 \boxplus \beta_2 = \mathsf{Enc}(m_1 m_2 - r_1 r_2)$.
5. Output $(b \cdot (m_1 - r_1) + c \cdot (m_2 - r_2)) \oplus (a \otimes \gamma)$.

Observe that the above technique can be generalized to compute degree-2 functions over an arbitrary number of inputs (Section 3.6).

We define and show a security property of this technique, which we refer to as *masked vs random* security.

Definition 5. *Let $\mathcal{H}$ be a linearly-homomorphic encryption scheme. For algorithm $\mathcal{A}$, consider the* masked vs random *experiment defined as:*

$$\mathsf{Exp}^{\mathsf{Msk}vs\mathsf{Rnd}}_{\mathcal{H},\mathcal{A}}(\kappa) \coloneqq \begin{cases} (\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa) \\ x \leftarrow \mathcal{A}(\mathsf{pk}) \\ y_0,\, r \xleftarrow{\$} \mathcal{M};\;\; y_1 \coloneqq x - r \\ b \xleftarrow{\$} \{0,1\} \\ b' \leftarrow \mathcal{A}(y_b,\, \mathsf{Enc}(\mathsf{pk},r)) \\ \textit{if } b = b' \textit{ output } 1\textit{, otherwise, } 0. \end{cases}$$

The advantage function of $\mathcal{A}$ in the masked vs random security game is defined as:

$$\mathsf{Adv}^{\mathsf{Msk}vs\mathsf{Rnd}}_{\mathcal{H},\mathcal{A}}(\kappa) \coloneqq \left|\Pr[\mathsf{Exp}^{\mathsf{Msk}vs\mathsf{Rnd}}_{\mathcal{H},\mathcal{A}}(\kappa) = 1] - \tfrac{1}{2}\right| \ .$$

Lemma 1. *Let $\mathcal{H}$ be a linearly-homomorphic encryption scheme. For every p.p.t. adversary $\mathcal{A}$, there exists a p.p.t. adversary $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\mathsf{Msk}vs\mathsf{Rnd}}_{\mathcal{H},\mathcal{A}}(\kappa) \le \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{H},\mathcal{B}}(\kappa) \ .$$

We refer to Appendix A for a proof.

3.5 Distributed Threshold Encryption

In our constructions, we use a variant of public key encryption in which a party can encrypt a message to a set of public keys in such a way that all the corresponding secret keys need to be used in decryption. This is essentially a version of n-out-of-n distributed threshold encryption (introduced by De Santis et al. [25] and further formalized by Cortier et al., [22]) with three main functional differences: each party generates its own key pair independently (i.e., there is no dealer or interactive protocol for key generation); a common group public key can be constructed on-the-fly (i.e., one can dynamically pick the set of recipients); decryption can be performed by applying the secret keys in a simple sequential fashion. Finally, we also require this scheme be equipped with an algorithm AddPk that allows one to add a recipient's public key (with knowledge of its secret key) to an existing ciphertext. A formalization follows.

Definition 6 (Distributed Threshold Encryption)**.** *A* distributed threshold encryption *scheme is a tuple of four efficiently computable algorithms:*

- $\mathsf{KeyGen}(1^\kappa)$*: this protocol is independently run by each party* P_i *(*$i \in [n]$ *and* $n \in \mathbb{N}$*); on input security parameter* κ*, output its own key pair* $(\mathsf{sk}_i, \mathsf{pk}_i)$
- $\mathsf{Enc}(\mathsf{pk}_1, \ldots, \mathsf{pk}_n; m)$*: on input the set of all public keys* $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ *and a message* m*, outputs a ciphertext* ct*.*
- $\mathsf{AddPk}(\mathsf{sk}, \mathsf{ct})$*: on input a secret key* sk *and a ciphertext* ct*, outputs a ciphertext* ct'*.*
- $\mathsf{ShareDec}(\mathsf{sk_i}, \mathsf{ct})$*: this is a share decryption algorithm that on input a secret key* sk_i *and a ciphertext* ct*, outputs a decryption share* $\mathsf{c_i}$*.*
- $\mathsf{Rec}(\mathsf{c}_1, \ldots, \mathsf{c_t})$*: this is the reconstruction algorithm that, given a set of decryption shares* $\mathsf{c}_1, \ldots, \mathsf{c_t}$ *outputs value.*

A distributed threshold encryption scheme is said *correct* if for every $m \in \mathcal{M}$, the following probability (over the coins of KeyGen and Enc) is overwhelming in κ:

$$\Pr\left[\mathsf{Rec}(\{\mathsf{ShareDec}(\mathsf{sk_i},\ \mathsf{Enc}(\mathsf{pk_1}, \ldots, \mathsf{pk_n}; \mathsf{m}))\}_{\mathsf{i}\in[\mathsf{n}]}) = \mathsf{m}\right].$$

Furthermore AddPk is such that for n honestly generated key pairs $\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in[n]}$ and every $m \in \mathcal{M}$, it holds $\mathsf{AddPk}(\mathsf{sk}_n, \mathsf{Enc}(\mathsf{pk}_1, \ldots, \mathsf{pk}_{n-1}; m)) = \mathsf{Enc}(\mathsf{pk}_1, \ldots, \mathsf{pk}_n; m)$.

For security, we define IND-CPA security against static corruptions. This is essentially classical IND-CPA security with the difference that here the adversary receives one honestly generated public key pk_1, specifies two messages m_0 and m_1 and up to $n-1$ public keys $\mathsf{pk}_2, \ldots, \mathsf{pk}_n$ (of which it can know the secret keys), then it receives $\mathsf{Enc}(\mathsf{pk}_1, \ldots, \mathsf{pk}_n; m_b)$ for a random b and must return a bit b'.

In our work we are interested in distributed threshold public key encryption schemes that are linearly-homomorphic. An efficient instantiation of it can be obtained based on ElGamal as follows. (A similar scheme can be found in [32].) Assume that a prime order group $\mathbb{G}$ (with generator g) is common to all parties. During KeyGen each party creates an ElGamal key pair $(\mathsf{pk}, \mathsf{sk})$ such that $\mathsf{pk} = g^{\mathsf{sk}}$ for a random sk. Given public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$, an encryption of m is $\mathsf{ct} = (a, b)$, computed as $a = g^r, b = m \prod_{i\in[n]} \mathsf{pk}_i^r$ for a randomly chosen r. Algorithm $\mathsf{AddPk}(\mathsf{sk}, \mathsf{ct})$ returns $(a, b \cdot a^{\mathsf{sk}})$. Algorithm $\mathsf{ShareDec}$ on ciphertext ct and secret key share sk_i returns $\mathsf{c_i} = \mathsf{a}^{\mathsf{sk_i}}$. Finally, algorithm Rec outputs $b / \prod_{i\in[n]} \mathsf{c_i}$.

Note that, at encryption time, one could pick any subset of the public keys, say $\{\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_t}\} \subseteq \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n, \}$, and let $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_t}; m)$. Also, let I be the set of indices $i_1, \ldots, i_t$. Thus, the reconstruction algorithm recovers m only if it is given in input the decryption shares $\{\mathsf{c_i} \leftarrow \mathsf{Dec}(\mathsf{sk_i}, \mathsf{ct})\}_{\mathsf{i}\in\mathsf{I}}$.

Correctness and security of this construction are rather obvious and we omit them.

3.6 Quadratic Functions

Definition 7 (Arithmetic circuit)**.** *An* arithmetic circuit *is a directed acyclic graph with* ℓ *sources (nodes with in-degree* 0*) and other nodes of in-degree* 2*, called internal gates. Sources are labeled by inputs* $x_1, \ldots, x_\ell$*, all other nodes are labeled by* $+$ *or* $*$*.*

We consider circuits with a single output node or sink (of out-degree 0). A circuit f of this form computes an arithmetic function over a set R with two operations, $+$ and $*$, defined over it (usually forming a ring), from $R^\ell \to R$. More concretely, after assigning a value $x_i \in R$ to the input nodes, the internal gates can be evaluated in any order consistent with the underlying graph, resulting in a unique output value, denoted by $eval_f(\boldsymbol{x})$.

Inputs of a circuit that have a fixed associated value are called *constants*, otherwise they are called *linear terms* or *variables*. We say a circuit is *quadratic* if it computes a quadratic function over R. More precisely, a circuit is quadratic if every node *can be tagged* as 0 (con-

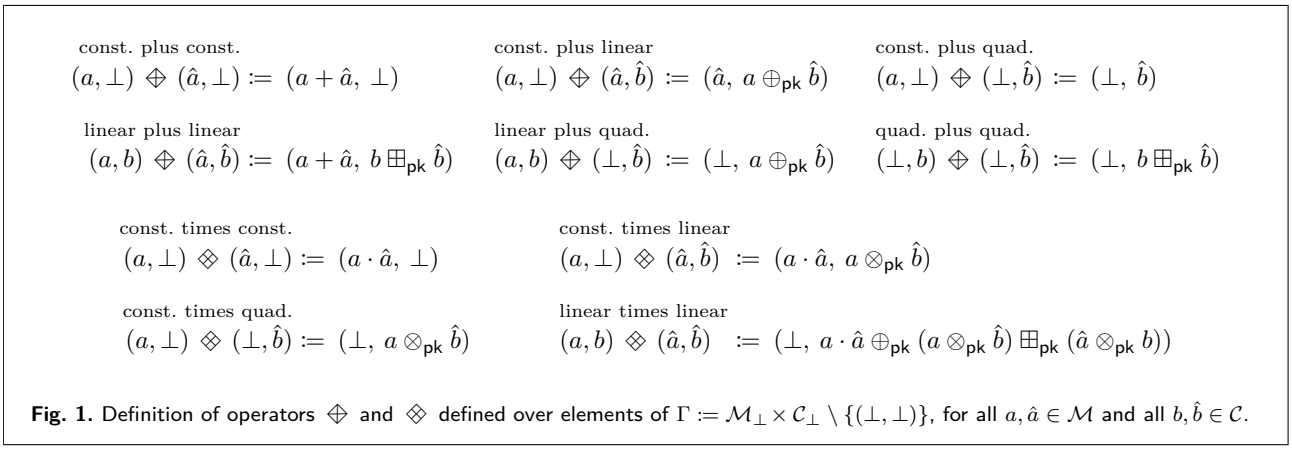

Fig. 1. Definition of operators $\oplus\!\!\!\!\oplus$ and $\otimes\!\!\!\!\otimes$ defined over elements of $\Gamma := \mathcal{M}_\perp \times \mathcal{C}_\perp \setminus \{(\perp,\perp)\}$, for all $a,\hat{a} \in \mathcal{M}$ and all $b,\hat{b} \in \mathcal{C}$.

stant), 1 (linear) or 2 (quadratic) preserving the natural hierarchy of the circuit. Namely, constant source nodes are tagged with 0, variable source nodes are tagged with 1 and for every non-source node u labeled with op and with predecessors v_1 and v_2, it holds:

$$tag(u) = \max\{tag(v_1), tag(v_2)\} \qquad \text{if } op = +$$
$$tag(u) = tag(v_1) + tag(v_2) \qquad \text{if } op = * \ .$$

We now describe a method to homomorphically evaluate a quadratic circuit, that leverages a linearly-homomorphic encryption scheme, say with message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$. Let $\mathcal{M}_\perp$ and $\mathcal{C}_\perp$ be defined as $\{\perp\} \cup \mathcal{M}$ and $\{\perp\} \cup \mathcal{C}$ respectively. The method is based on the technique of [10] (see Section 3.4), with a small extension to deal with additions by constant (not supported in [10]). In Figure 1, we define two operations, an homomorphic addition $\oplus\!\!\!\!\oplus$ and an homomorphic multiplication $\otimes\!\!\!\!\otimes$, over the set $\Gamma := \mathcal{M}_\perp \times \mathcal{C}_\perp \setminus \{(\perp,\perp)\}$. Elements in $\mathcal{M} \times \{\perp\}$ represent *constants*, elements in $\mathcal{M} \times \mathcal{C}$ represent *linear terms*, whereas elements in $\{\perp\} \times \mathcal{C}$ represent *quadratic terms*.

Observe that although operators $\oplus\!\!\!\!\oplus$ and $\otimes\!\!\!\!\otimes$ may be probabilistic, since they are defined based on the homomorphic operators $\oplus$, $\otimes$ and $\boxplus$, they inherit associativity, commutativity and distributivity from these operations (as well as from $+, *$ in $\mathcal{M}$).[3] Also, observe that operator $\otimes\!\!\!\!\otimes$ is not defined between a linear and a quadratic term or between two quadratic terms. This is not necessary to evaluate quadratic circuits.

In order to state our next lemma, we define the decryption of elements in Γ as follows, for every $a \in \mathcal{M}$ and every $b \in \mathcal{C}$:

$$\mathsf{Dec}_{\mathsf{sk}}((a,b)) := a \qquad \mathsf{Dec}_{\mathsf{sk}}((\perp,b)) := \mathsf{Dec}_{\mathsf{sk}}(b)$$

3 These properties hold with respect to fixed randomness.

Lemma 2. *Let f be a non-constant quadratic circuit with variables x_i, $\forall i \in [\ell]$ and let $\boldsymbol{x}, \boldsymbol{r} \in \mathcal{M}^\ell$. It holds:*

$$\mathsf{Dec}_{\mathsf{sk}}\left(eval_f(\boldsymbol{x}-\boldsymbol{r},\ \mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{r}))\right) = f(\boldsymbol{x}) - f(\boldsymbol{r}) \ ,$$

where the evaluation of f takes place over set Γ with operations $\oplus\!\!\!\!\oplus$ for addition and $\otimes\!\!\!\!\otimes$ for multiplication.

We refer to Appendix A for a proof.

Context Hiding. We claim that the result of $eval_f$ contains no information about the original plaintexts, what we define as *context hiding*. We could formalize this by showing that there exists a simulator $\widehat{\mathsf{Sim}}$ that on input f, $f(\boldsymbol{x})$, the random vector $\boldsymbol{r}$ and the secret key, can perfectly simulate the output of $eval_f$. This can be done based on the perfect circuit privacy (Definition 4) of $\mathcal{H}$. A formal proof of the context hiding of $eval_f$ could be obtained in essentially the same way as in [10].

4 Multi-Authority C-FE

Controlled Functional Encryption. We start by providing an overview of Controlled Functional Encryption as defined in [37]. In a C-FE scenario, there are three types of entities denoted as *data-owners*, *clients*, and a *central authority*. The authority runs a Setup algorithm to produce public system parameters and its own secret key. A data-owner uses an encryption routine Enc to encrypt its data item x. The resulting ciphertext ct is sent to clients and may also include an arbitrary policy λ. A client is the party interested in computing $f(x)$, given ciphertext ct. To do so, the client first issues a *key request* (with references to ct and the function f to be computed) to the authority. Via the key request, the authority obtains

the policy λ (associated to ct), which can be evaluated (using an arbitrary logic) to decide whether to grant or deny the client's request. If the request is to be granted, the authority issues a functional key sk^f to the client. Note that sk^f is only valid to compute f over the data encrypted under ct. Finally the client runs a decryption algorithm Dec with input sk^f and ct to learn $f(x)$.

Correctness informally states that, if all parties are honest and execute the algorithms correctly, the authority obtains the policy λ attached to the ciphertext by the data-owner, while the client learns $f(x)$. Security guarantees that, unless the client and the authority collude, (i) no information on x is leaked to the authority, and (ii) nothing but $f(x)$ is learned by the client.

As discussed above, the authors of [37] instantiate a very efficient C-FE scheme for single-input, linear functions, based on a CCA2 encryption scheme. They also build a C-FE for arbitrary functions (again, single-input) that leverages garbled circuits, making it particularly suited for functions over Boolean inputs.

Multi-Authority Controlled Functional Encryption. We extend the work of [37] to scenarios with *multiple authorities*. In our setting, a client must issue key requests to each authority and will only learn $f(x)$ if all of them decide to grant such requests. Hence, multi-authority C-FE considerably strengthens the security of C-FE. In particular, our security definitions ensure that no information on the data produced by owners is leaked (apart from $f(x)$, that is learned by the client), unless the client and *all* the authorities collude. In other words, as long as one authority is honest, data-owners are assured on the privacy of their data. We also consider functions with *multiple inputs* coming from possibly different data-owners. In Section 5.2, we provide an instantiation of mC-FE for quadratic functions using a CCA2 encryption scheme and a linearly-homomorphic encryption scheme.

Definition 8 (Multi-authority C-FE)**.** *Let $\mathcal{F}=\{\mathcal{F}_\kappa\}_{\kappa\in\mathbb{N}}$ be a function family, where $\mathcal{F}_\kappa$ is a collection of functions $\{F:\mathcal{X}_\kappa\times\Lambda_\kappa\to\mathcal{Y}_\kappa\}$ and let $n\in\mathbb{N}$. A* multi-authority C-FE *scheme for $\mathcal{F}$ consists of two distributed protocols $\{\mathsf{Auth\text{-}Setup}, \mathsf{KeyReq}\}$ and a tuple of efficiently computable algorithms $\{\mathsf{Cli\text{-}Setup}, \mathsf{Enc}, \mathsf{Dec}\}$:*

- $\mathsf{Auth\text{-}Setup}(1^\kappa)\to\{(\mathsf{mpk},\mathsf{msk}_i)\}_{i\in[n]}$*: for certain security parameter κ, the authorities $\mathsf{A}_1,\ldots,\mathsf{A}_n$ run a distributed protocol, where all obtain the master public key mpk (which will be distributed to all data-owners), while the k-th share of the master secret key msk_k is only obtained by the k-th authority A_k.*
- $\mathsf{Cli\text{-}Setup}(1^\kappa)\to(\mathsf{pk_c},\mathsf{sk_c})$*: on input the security parameter, outputs a pair of keys for the client.*
- $\mathsf{Enc}(\mathsf{mpk},\mathsf{pk_c},x,\lambda)\to\mathsf{ct}$*: on input a master public key, a client's public key , a value x, and a policy λ, outputs a ciphertext ct.*
- $\mathsf{KeyReq}((f,\mathbf{ct},\mathsf{pk_c});\{\mathsf{msk}_i\}_{i\in[n]})\to(\mathsf{sk}^f;\{\boldsymbol{\lambda}_i\}_{i\in[n]})$*: the client and the n authorities run a distributed protocol, where the client inputs a function f of arity ℓ and a vector of ℓ ciphertexts $\mathbf{ct}=(\mathsf{ct}_1,\ldots,\mathsf{ct}_\ell)$ together with their own public key; and every authority inputs their corresponding master secret key. As the outcome of this protocol, all authorities get a vector of policies associated to the requested ciphertexts (based on which they may decide to abort their execution) and the client receives a secret function key sk^f.*
- $\mathsf{Dec}(f,\mathbf{ct},\mathsf{sk_c},\mathsf{sk}^f)\to y$*: on input a function of arity ℓ, a list of ℓ ciphertexts, a client's secret key and a function key, outputs a value y.*

Remark. For generality, we have defined $\mathsf{Auth\text{-}Setup}$ and KeyReq as distributed protocols. In the spirit of functional encryption, we require protocol KeyReq be such that the authorities do not exchange messages between them and every authority interacts only once with the client, i.e., it receives the request from the client and answers back or aborts (if it decides not to grant the computation). Ideally, the $\mathsf{Auth\text{-}Setup}$ would require no interaction between the authorities so that new authorities can enter the system at any moment (our instantiation from Section 5 achieves this property).

Remark. Although requiring a client's public key for encryption seems to depart from the original notion of C-FE, we note that the C-FE model of [37] implicitly assumes the existence of a private communication channel between the data-owner and the client. Thus having a public key for every client is a way to realize such channel. Furthermore, we argue that it is reasonable to assume that data-owners (and authorities) are aware of client's identities (and their public keys) in a real deployment of C-FE. Indeed, if data-owners are to define *who* can use their data and authorities are to enforce those policies, well-known client identities are necessary.

We require a multi-authority C-FE be *correct*. Roughly, if all algorithms are executed correctly, after KeyReq the authorities recover the original policies set by the data-owners who produced the ciphertexts (this is for the authority to evaluate the data-owner's policies correctly). Further, after decryption, the client receives $f(x_1,\ldots,x_\ell)$ if all authorities granted the evaluation. In

Ideal World

- SETUP. Let $\mathcal{O}$ be an ideal functionality interacting with the data-owners, the clients and the n-authorities. $\mathcal{O}$ keeps tracks of ciphertexts and evaluation requests with tables T_1 and T_2, respectively. All tables are initially empty. Let also $\beta_i()$ be a Boolean function that models the evaluation of the policies set by data-owners as carried out by the i-th authority.
- DATA UPLOAD. A data-owner sends (x, C, λ) to $\mathcal{O}$, who creates a fresh handle h and sends it to the simulator. If the latter allows, then $\mathcal{O}$ stores the record $(h, x, \mathsf{C}, \lambda)$ in T_1, sends h to C and stops. Otherwise, $\mathcal{O}$ drops (x, C, λ) and stops.
- EVALUATION REQUEST. A client C sends f and $\boldsymbol{h} = (h_{i_1}, \ldots, h_{i_\ell})$ to $\mathcal{O}$. For each $j \in \{i_1, \ldots, i_\ell\}$, $\mathcal{O}$ checks that $(h_j, x_j, \mathsf{C}, \lambda_j) \in T_1$ for some x_j and λ_j; if the check fails for some j, $\mathcal{O}$ sends $\perp$ to the client and stops. Otherwise, let $\boldsymbol{\lambda} = (\lambda_{i_1}, \ldots, \lambda_{i_\ell})$. Oracle $\mathcal{O}$ draws a fresh handle t (the session ID) and, for $i \in [n]$ behaves as follows. If the i-th authority is corrupted, it sends $(t, f, \mathsf{C}, \boldsymbol{h}, \boldsymbol{\lambda}, i)$ to the simulator that may reply with a Boolean $allow_i$; otherwise, it sets $allow_i \coloneqq \beta_i(\boldsymbol{\lambda})$. If $allow_i$ is defined, then $\mathcal{O}$ sends $(t, i, allow_i)$ to C. If $allow_i$ equals true for all $i \in [n]$, then $\mathcal{O}$ stores $(t, f, \mathsf{C}, \boldsymbol{h})$ in T_2 and stops.
- DECRYPTION. A client C sends t to $\mathcal{O}$. If exists $(t, f', \mathsf{C}, \boldsymbol{h}') \in T_2$ for any f' and any $\boldsymbol{h}'$, then oracle $\mathcal{O}$ retrieves $(h_i, x_i, *, *)$ for each $h_i \in \boldsymbol{h}'$, computes $y = f(x_1, \ldots, x_\ell)$ and sends y to C. Otherwise, oracle $\mathcal{O}$ sends $\perp$ to C and stops.

Real World

- SETUP. Authorities run $\mathsf{Auth\text{-}Setup}$ with security parameter κ, and each gets $(\mathsf{mpk}, \mathsf{msk}_i)$, for $i \in [n]$. They distribute mpk among all other parties. Also, each client runs $\mathsf{Cli\text{-}Setup}$ (again, with security parameter κ) to obtain $(\mathsf{pk_c}, \mathsf{sk_c})$; $\mathsf{pk_c}$ is distributed to all other parties.
- DATA UPLOAD. A data-owner runs Enc on $(\mathsf{mpk}, \mathsf{pk_c}, x, \lambda)$ to get a ciphertext ct and sends it to all clients.
- EVALUATION REQUEST. A client C wishing to compute f over $\mathbf{ct}$ engages with the n authorities in $\mathsf{KeyReq}((f, \mathbf{ct}); \{\mathsf{msk}_i\}_{i\in[n]}) \to (\mathsf{sk}^f; \{\boldsymbol{\lambda}_i\}_{i\in[n]})$ and obtains sk^f if all authorities grant the computation.
- DECRYPTION. A client C runs Dec on input $(f, \mathbf{ct}, \mathsf{sk_c}, \mathsf{sk}^f)$ to obtain y.

Fig. 2. Ideal and real worlds for the security of multi-authority C-FE.

the following definition, we abuse notation to accommodate for vectors of data items.

Definition 9 (Correctness)**.** *A multi-authority C-FE scheme is said to be correct if there exists an overwhelming function, δ such that, for all (sufficiently large) $\kappa \in \mathbb{N}$, $f \in \mathcal{F}_\kappa$, and all $\boldsymbol{x} = (x_1, \ldots, x_\ell) \in \mathcal{X}_\kappa$, $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_\ell) \in \Lambda_\kappa$, the following probability is $\geq \delta(\kappa)$:*

$$\Pr\left[\begin{array}{c} \{(\mathsf{mpk}, \mathsf{msk}_i)\}_{i\in[n]} \leftarrow \mathsf{Auth\text{-}Setup}(1^\kappa) \\ (\mathsf{pk_c}, \mathsf{sk_c}) \leftarrow \mathsf{Cli\text{-}Setup}(1^\kappa) \\ \mathbf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{pk_c}, \boldsymbol{x}, \boldsymbol{\lambda}) \\ (\mathsf{sk}^f, \{\boldsymbol{\lambda}'_i\}_{i\in[n]}) \leftarrow \mathsf{KeyReq}((f, \mathbf{ct}, \mathsf{pk_c}); \{\mathsf{msk}_i\}_{i\in[n]}) \\ y \leftarrow \mathsf{Dec}(f, \mathbf{ct}, \mathsf{sk_c}, \mathsf{sk}^f) \\ \\ y = f(\boldsymbol{x}) \ \wedge \ \forall i \in [n], \ \boldsymbol{\lambda}'_i = \boldsymbol{\lambda} \end{array}\right]$$

where the probability is taken over the coins of the algorithms that are involved.

Regarding security, our goal is to ensure that a client can learn the output of a specific function over a given (set of) ciphertext(s) only if all of the authorities allow so; furthermore no additional information can be leaked about the encrypted data, unless the client and all of the authorities collude. We express such security requirements using the simulation paradigm. Namely, we define an *ideal functionality* that acts as a trusted third party and captures the desired security guarantees, i.e., the description of what every party in the system (data-owners, clients and the authorities) should know and learn from the interaction with each other. The methodology consists of defining an *ideal world* (based

on the ideal functionality) and a *real world* (based on the C-FE construction) and proving that all attacks in the real world are inherent. Namely, every attack in the real world can also be performed in the simulated world, which is formalized by showing that for every algorithm controlling a set of parties in the real world, there exists another algorithm (simulator) controlling the same set of parties in the ideal world that produces an indistinguishable output. We refer to Figure 2 for a description of the two worlds in the case of multi-authority C-FE.

Definition 10 (Security of multi-authority C-FE)**.** *A multi-authority C-FE scheme is said to be secure if for every adversary $\mathcal{A}$ in the* real world *(actively corrupting a subset of clients and possibly up to $n-1$ authorities; or corrupting all n authorities, but no client), there exists a simulator $\mathcal{S}$ in the* ideal world *(corrupting the same set of parties) who produces an output that is indistinguishable from the output of $\mathcal{A}$.*

5 mC-FE for Quadratic Functions

In this section we provide a construction of a multi-authority C-FE for quadratic functions.

5.1 Single-Authority C-FE

We start by defining a very efficient scheme in the case of a single authority. Our construction relies on the method used for building *labeled homomorphic encryption* for quadratic polynomials [18] (see Section 3.6). In particular, the authors of [18] encrypt a message x as $\mathsf{e} = (a = x - r, b = \mathsf{Enc}(r))$ where Enc is a linear-y homomorphic and r is fresh randomness used to blind x. Given e, anybody can homomorphically compute $\mathsf{Enc}(f(x) - f(r))$ for a quadratic function f (we refer to our Section 3.6 for more details about this procedure called $eval_f$). Our idea is to let the client hold the decryption key of Enc but we wrap ciphertext e with an additional encryption layer by using a CCA2 encryption scheme and the authority's public key; further, we reveal the blinding r to the client. Hence the authority (and only the authority) can recover e and evaluate homomorphically $\mathsf{Enc}(f(x) - f(r))$; this ciphertext is essentially the function key; it is given to the client that uses its secret key to recover the plaintext $f(x) - f(r)$, and removes $f(r)$ to obtain $f(x)$. This basic idea can be easily extended to accommodate for functions of any arity where inputs are provided by multiple data-owners. Further, the CCA2 encryption scheme allows data-owners to attach an arbitrary policy to their ciphertexts, ensuring that it cannot be tampered.

- $\Upsilon.\mathsf{Auth\text{-}Setup}(1^\kappa)$: on input κ,
  1. run $\mathcal{E}.\mathsf{KeyGen}(1^\kappa)$ to get $(\mathsf{pk}^\mathcal{E}, \mathsf{sk}^\mathcal{E})$,
  2. set $\mathsf{mpk} := \mathsf{pk}^\mathcal{E}$, and $\mathsf{msk} := \mathsf{sk}^\mathcal{E}$,
  3. output $(\mathsf{mpk}, \mathsf{msk})$.
- $\Upsilon.\mathsf{Cli\text{-}Setup}(1^\kappa)$: on input the security parameter,
  1. run $\mathcal{H}.\mathsf{KeyGen}(1^\kappa)$ to get $(\mathsf{pk}^\mathcal{H}_\mathsf{c}, \mathsf{sk}^\mathcal{H}_\mathsf{c})$,
  2. output $(\mathsf{pk}^\mathcal{H}_\mathsf{c}, \mathsf{sk}^\mathcal{H}_\mathsf{c})$.
- $\Upsilon.\mathsf{Enc}(\mathsf{mpk}, \mathsf{pk}^\mathcal{H}_\mathsf{c}, x, \lambda)$: on input $\mathsf{mpk} = \mathsf{pk}^\mathcal{E}$, the client's public key $\mathsf{pk}^\mathcal{H}_\mathsf{c}$, $x \in \mathcal{M}$ and policy λ,
  1. sample $r \xleftarrow{\$} \mathcal{M}$,
  2. compute $a := x - r$ and $b := \mathcal{H}.\mathsf{Enc}(\mathsf{pk}^\mathcal{H}_\mathsf{c}, r)$,
  3. set $\mathsf{ct} := (r,\ \mathcal{E}.\mathsf{Enc}(\mathsf{pk}^\mathcal{E}, (a, b, \lambda)))$,
  4. output ct.
- $\Upsilon.\mathsf{KeyReq}((f, \mathbf{ct}, \mathsf{pk}^\mathcal{H}_\mathsf{c}); \mathsf{msk})$: where $\mathbf{ct}$ includes ℓ ciphertexts,
  1. the client parses ct_j as (r_j, e_j), $\forall j \in [\ell]$,
  2. sends $(f, \{\mathsf{e}_j\}_{j\in[\ell]})$ to the authority,
  3. who sets $(a_j, b_j, \lambda_j) := \mathcal{E}.\mathsf{Dec}(\mathsf{sk}^\mathcal{E}, \mathsf{e}_j)$, $\forall j \in [\ell]$
  4. then sets $\boldsymbol{a} = (a_1, \ldots, a_\ell)$ ($\boldsymbol{b}$, $\boldsymbol{\lambda}$ analogously),
  5. if $\exists j \in [\ell] : a_j \notin \mathcal{M} \vee b_j \notin \mathcal{C} \vee \lambda_j \notin \Lambda$, it sets $\mathsf{sk}^f := \perp$, otherwise, $\mathsf{sk}^f := eval_f(\boldsymbol{a}, \boldsymbol{b})$ (using $\mathsf{pk}^\mathcal{H}_\mathsf{c}$),
  6. the authority stores $\boldsymbol{\lambda}$, sending sk^f to the client.
- $\Upsilon.\mathsf{Dec}(f, (\mathsf{ct}_1 \ldots, \mathsf{ct}_\ell), \mathsf{sk}^\mathcal{H}_\mathsf{c}, \mathsf{sk}^f)$:
  1. parse ct_j as (r_j, e_j), for every $j \in [\ell]$,
  2. output $\mathcal{H}.\mathsf{Dec}(\mathsf{sk}^\mathcal{H}_\mathsf{c}, \mathsf{sk}^f) + f(r_1, \ldots, r_\ell)$.

Fig. 3. Instantiation of a single-authority C-FE for quadratic functions, based on a linearly-homomorphic encryption scheme $\mathcal{H}$, and a CCA2 public-key encryption scheme $\mathcal{E}$.

Figure 3 provides the details of our instantiation. In a bit more of detail, during $\Upsilon.\mathsf{Auth\text{-}Setup}$ the authority creates secret and public parameters that are essentially the secret and public key $\mathsf{pk}^\mathcal{E}, \mathsf{sk}^\mathcal{E}$ for a CCA2 encryption scheme $\mathcal{E}$. Similarly, each client runs $\Upsilon.\mathsf{Cli\text{-}Setup}$ to compute a key pair $\mathsf{pk}^\mathcal{H}_\mathsf{c}, \mathsf{sk}^\mathcal{H}_\mathsf{c}$ for a linearly-homomorphic encryption scheme $\mathcal{H}$.

The encryption routine $\Upsilon.\mathsf{Enc}$ can be used to encrypt message $x \in \mathcal{M}$ for a client with public key $\mathsf{pk}^\mathcal{H}_\mathsf{c}$, picking a fresh $r \in \mathcal{M}$ at random and computing $\mathsf{ct} = (r, \mathsf{e})$ where $\mathsf{e} = \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}^\mathcal{E}}(x - r, \mathcal{H}.\mathsf{Enc}_{\mathsf{pk}^\mathcal{H}_\mathsf{c}}(r))$. The CCA2 ciphertext e should also carry the policy λ, which we omit here for simplicity.

When a client wants to compute a function f of arity ℓ on ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell$, it engages in protocol $\Upsilon.\mathsf{KeyReq}$ with the authority. Let $\mathsf{ct}_i = (r_i, \mathsf{e}_i)$, for every $i \in [\ell]$, and defines $\boldsymbol{r} = r_1, \ldots, r_\ell$, $\mathbf{e} = \mathsf{e}_1, \ldots, \mathsf{e}_\ell$. The authority decrypts $\mathbf{e}$ (by using $\mathsf{sk}^{\mathcal{E}}$) component-wise to get $(\boldsymbol{x}-\boldsymbol{r}, \mathcal{H}.\mathsf{Enc}_{\mathsf{pk}^{\mathcal{H}}_{\mathsf{c}}}(\boldsymbol{r}))$. (The authority also gets the policies as defined by data-owners and uses them to decide whether to process the client's request or to abort.) Next, the authority $eval_f$ to compute $\mathcal{H}.\mathsf{Enc}_{\mathsf{pk}^{\mathcal{H}}_{\mathsf{c}}}(f(\boldsymbol{x})-f(\boldsymbol{r}))$, and sends it to the client. Finally, the client runs $\Upsilon.\mathsf{Dec}$ that decrypts the received ciphertext using $\mathsf{sk}^{\mathcal{H}}_{\mathsf{c}}$ and adds $f(\boldsymbol{r})$ to the result, getting $f(\boldsymbol{x})$.

Theorem 1 (Correctness)**.** *The scheme from Figure 3 is a* correct *1-authority C-FE scheme.*

Proof. Let f be a function, $\boldsymbol{x} \in \mathcal{M}^\ell$, and policies $\boldsymbol{\lambda}$. Run:

$$(\mathsf{pk}^{\mathcal{E}}, \mathsf{sk}^{\mathcal{E}}) \leftarrow \Upsilon.\mathsf{Auth\text{-}Setup}(1^\kappa) \quad (\mathsf{pk}^{\mathcal{H}}_{\mathsf{c}}, \mathsf{sk}^{\mathcal{H}}_{\mathsf{c}}) \leftarrow \mathsf{Cli\text{-}Setup}(1^\kappa)$$

Then run $(\boldsymbol{r}, \mathbf{e}) \leftarrow \Upsilon.\mathsf{Enc}(\mathsf{pk}^{\mathcal{E}}, \boldsymbol{x}, \boldsymbol{\lambda}, \mathsf{pk}^{\mathcal{H}}_{\mathsf{c}})$. Notice that computing $\Upsilon.\mathsf{KeyReq}((f, \mathbf{ct}, \mathsf{pk}^{\mathcal{H}}_{\mathsf{c}}); \mathsf{msk})$ will result in $(\mathsf{sk}^f; \boldsymbol{\lambda}')$ where $\mathsf{sk}^f = eval_f(\boldsymbol{x}-\boldsymbol{r}, \mathcal{H}.\mathsf{Enc}_{\mathsf{pk}^{\mathcal{H}}_{\mathsf{c}}}(\boldsymbol{r}))$ and $\boldsymbol{\lambda}' = \boldsymbol{\lambda}$. Now, $\Upsilon.\mathsf{Dec}(f, \mathbf{ct}, \mathsf{sk}^f, \mathsf{sk}^{\mathcal{H}}_{\mathsf{c}})$ produces $\mathcal{H}.\mathsf{Dec}(\mathsf{sk}^{\mathcal{H}}_{\mathsf{c}}, \mathsf{sk}^f) + f(\boldsymbol{r})$, that (by Lemma 2) equals $f(\boldsymbol{x})$. □

Theorem 2 (Hbc authority)**.** *Let $\mathcal{H}$ be a linearly-homomorphic encryption scheme and let $\mathcal{E}$ be a CCA2 public-key encryption scheme. The scheme from Figure 3 is such that for every adversary $\mathcal{A}$ in the* real world *(passively corrupting the authority), there exists a simulator in the* ideal world *(also corrupting the authority) who produces an output that is indistinguishable from the one of $\mathcal{A}$.*

Theorem 3 (Malicious clients)**.** *Let $\mathcal{H}$ be a linearly-homomorphic encryption scheme and let $\mathcal{E}$ be a CCA2 public-key encryption scheme. The scheme from Figure 3 is such that for every adversary $\mathcal{A}$ in the* real world *actively corrupting a subset of clients, there exists a simulator in the* ideal world *(corrupting the same set of parties) who produces an output that is indistinguishable from the one of $\mathcal{A}$.*

We refer to Appendix A for a proof of both Theorems.

- $\Psi.\mathsf{Auth\text{-}Setup}(1^\kappa)$: on input κ,
  1. authority $i \in [n]$ runs $(\mathsf{pk}^{\mathcal{H}}_i, \mathsf{sk}^{\mathcal{H}}_i) \leftarrow \mathcal{H}.\mathsf{KeyGen}(1^\kappa)$,
  2. authority $i \in [n]$ runs $(\mathsf{pk}^{\mathcal{E}}_i, \mathsf{sk}^{\mathcal{E}}_i) \leftarrow \mathcal{E}.\mathsf{KeyGen}(1^\kappa)$,
  3. set $\mathsf{mpk} := (\{(\mathsf{pk}^{\mathcal{H}}_i, \mathsf{pk}^{\mathcal{E}}_i)\}_{i\in[n]})$,
  4. set $\mathsf{msk}_i := (\mathsf{sk}^{\mathcal{H}}_i, \mathsf{sk}^{\mathcal{E}}_i)$ for all $i \in [n]$,
  5. the output is $((\mathsf{mpk}, \mathsf{msk}_1); \ldots; (\mathsf{mpk}, \mathsf{msk}_n))$.
- $\Psi.\mathsf{Cli\text{-}Setup}(1^\kappa)$: on input κ,
  1. run $\mathcal{H}.\mathsf{KeyGen}(1^\kappa)$ to get $(\mathsf{pk}^{\mathcal{H}}_{\mathsf{c}}, \mathsf{sk}^{\mathcal{H}}_{\mathsf{c}})$,
  2. output $(\mathsf{pk}^{\mathcal{H}}_{\mathsf{c}}, \mathsf{sk}^{\mathcal{H}}_{\mathsf{c}})$.
- $\Psi.\mathsf{Enc}(\mathsf{mpk}, \mathsf{pk}^{\mathcal{H}}_{\mathsf{c}}, x, \lambda)$: on input the master public key $\mathsf{mpk} = \{(\mathsf{pk}^{\mathcal{H}}_i, \mathsf{pk}^{\mathcal{E}}_i)\}_{i\in[n]}$, $x \in \mathcal{M}$ and $\lambda \in \Lambda$,
  1. sample $r_i \xleftarrow{\$} \mathcal{M}$, for $i \in [n]$,
  2. set $r_0 := x$,
  3. set $\Omega_i := \{\mathsf{pk}^{\mathcal{H}}_{\mathsf{c}}, \mathsf{pk}^{\mathcal{H}}_i, \mathsf{pk}^{\mathcal{H}}_{i+1}, \ldots, \mathsf{pk}^{\mathcal{H}}_n\}$,
  4. let $a_i := r_{i-1} - r_i$, $b_i := \mathcal{H}.\mathsf{Enc}(\Omega_i, r_i)$, for $i \in [n]$,
  5. set $\mathsf{ct} := \left(r_n, \{\mathcal{E}.\mathsf{Enc}(\mathsf{pk}^{\mathcal{E}}_i, (a_i, b_i, \lambda))\}_{i\in[n]}\right)$,
  6. output ct.
- $\Psi.\mathsf{KeyReq}((f, \mathbf{ct}, \mathsf{pk}^{\mathcal{H}}_{\mathsf{c}}); \{\mathsf{msk}_i\}_i)$: where $\mathbf{ct}$ includes ℓ ciphertexts. The client sets $i = 1$. While $i \le n$:
  1. the client parses ct_j as $(r_{n,j}, (\mathsf{e}^{(1)}_j, \ldots, \mathsf{e}^{(n)}_j))$, $\forall j \in [\ell]$,
  2. if $i = 1$, it sends $(\mathsf{e}^{(1)}_1, \ldots, \mathsf{e}^{(1)}_\ell)$ to authority 1, if $i > 1$, $(\mathsf{sk}^f_{i-1}, (\mathsf{e}^{(i)}_1, \ldots, \mathsf{e}^{(i)}_\ell))$ to authority i,
  3. authority i parses msk_i as $(\mathsf{sk}^{\mathcal{H}}_i, \mathsf{sk}^{\mathcal{E}}_i)$,
  4. $\forall j \in [\ell]$, it sets $(a_j, b_j, \lambda_j) := \mathcal{E}.\mathsf{Dec}(\mathsf{sk}^{\mathcal{E}}_i, \mathsf{e}^{(i)}_j)$, and $\boldsymbol{a} = (a_1, \ldots, a_\ell)$, (define $\boldsymbol{b}$, $\boldsymbol{\lambda}$ analogously),
  5. if $i = 1$, it computes $\gamma_i := eval_f(\boldsymbol{a}, \boldsymbol{b})$, if $i > 1$, it computes $\gamma_i := \mathsf{sk}^f_{i-1} \boxplus_{\Omega_i} eval_f(\boldsymbol{a}, \boldsymbol{b})$,
  6. it sets $\mathsf{sk}^f_i := \mathcal{H}.\mathsf{Dec}(\mathsf{sk}^{\mathcal{H}}_i, \gamma_i)$,
  7. it stores $\boldsymbol{\lambda}$ and sends sk^f_i to the client,
  8. the client increments i and goes to Step 2.

  The client defines its secret key $\mathsf{sk}^f := \mathsf{sk}^f_n$.
- $\Psi.\mathsf{Dec}(f, (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell), \mathsf{sk}^{\mathcal{H}}_{\mathsf{c}}, \mathsf{sk}^f)$:
  1. parse ct_j as $(r_{n,j}, (\mathsf{e}^{(1)}_j, \ldots, \mathsf{e}^{(n)}_j))$, for all $j \in [\ell]$,
  2. output $\mathcal{H}.\mathsf{Dec}(\mathsf{sk}^{\mathcal{H}}_{\mathsf{c}}, \mathsf{sk}^f) + f(r_{n,1}, \ldots, r_{n,\ell})$.

Fig. 4. Construction of a multi-authority C-FE for quadratic functions, based on the multi-key linearly-homomorphic encryption scheme, $\mathcal{H}$, and the CCA2 public-key encryption scheme, $\mathcal{E}$.

5.2 Main Construction

In this section we show our mC-FE for quadratic functions. We design the scheme by extending the one for single-authority described in the previous section. In this case, however, we need to solve a number of technical challenges related to guaranteeing security when a client colludes with (up to) all but one authority, and avoiding collusion between authorities. We refer to the introduction for a discussion on these challenges.

A key idea of our solution is the use of secret-sharing of the encrypted message. More in details, in order to en-

crypt x, a data-owner picks n fresh blindings $r_1, \ldots, r_n$. Assume $r_0 = x$, the data-owner creates a "chain" of ciphertexts $\mathsf{e}^{(1)}, \ldots, \mathsf{e}^{(n)}$ so that $\mathsf{e}^{(i)}$ is the encryption of $(r_{i-1} - r_i, \mathsf{Enc}(r_i))$ under the public key of authority A_i, for $i \in [n]$. As before, Enc is the encryption routine of a linearly-homomorphic encryption scheme and the public key is the one of the client. On the other hand, the encryption scheme to create $\mathsf{e}^{(i)}$ is CCA2 and the public key is the one of authority A_i. Finally, the client also gets r_n from the data-owner.

Given $\mathsf{e}^{(i)}$, the i-th authority uses its secret key to recover $(r_{i-1} - r_i, \mathsf{Enc}(r_i))$, it homomorphically computes $\mathsf{Enc}(f(r_{i-1}) - f(r_i)))$, and returns it to the client. The latter decrypts the ciphertexts returned by the authorities and computes $\sum_{i=1}^{i=n} f(r_{i-1}) - f(r_i)$ that is equal to $f(r_0) - f(r_n)$. Finally, the client removes $f(r_n)$ and obtains $f(r_0)$ as desired (recall that $r_0 = x$).

The above design, however, has a security flaw. In particular, given $\mathsf{e}^{(i)} = (r_{i-1} - r_i, \mathsf{Enc}(r_i))$, a client that colludes with an authority can recover r_i and, therefore, r_{i-1}. For example, if the client colludes with authority A_1 they can recover r_0 (that is, x) from $\mathsf{e}^{(1)} = (r_0 - r_1, \mathsf{Enc}(r_1))$. We fix this flaw by using the notion of distributed threshold (linearly-homomorphic) encryption that we mentioned in Section 3.5. The idea is that to compute $\mathsf{Enc}(r_i)$ we use a set of public keys that include that of the client and the ones of authorities in the range $[n] \setminus [i-1]$ (therefore authorities also have public keys for the linearly-homomorphic encryption scheme).

In more detail, with $\mathsf{Auth\text{-}Setup}$ each authority generates a keypair for a linearly-homomorphic encryption scheme $\mathcal{H}$ and a CCA2 encryption scheme $\mathcal{E}$, whereas each client generates a keypair for $\mathcal{H}$ with $\mathsf{Cli\text{-}Setup}$.

The Enc routine encrypts a message x as follows. It draws n values $r_1, \ldots, r_n$ uniformly at random and sets $r_0 = x$. Next, it sets $\mathsf{ct} = \left(r_n, \{\mathsf{e}^{(i)}\}_{i \in [n]}\right)$ where $\mathsf{e}^{(i)}$ is a CCA2 encryption of $(r_{i-1} - r_i, \mathcal{H}.\mathsf{Enc}(\Omega_i, \boldsymbol{r}_i))$ under $\mathsf{pk}_i^{\mathcal{E}}$ and $\mathcal{H}.\mathsf{Enc}$ uses the following set of public keys $\Omega_i := \{\mathsf{pk}_{\mathsf{c}}^{\mathcal{H}}, \mathsf{pk}_i^{\mathcal{H}}, \mathsf{pk}_{i+1}^{\mathcal{H}}, \ldots, \mathsf{pk}_n^{\mathcal{H}}\}$, i.e., the public key $\mathsf{pk}_{\mathsf{c}}^{\mathcal{H}}$ of the client and the public keys of authorities $\mathsf{A}_i, \mathsf{A}_{i+1} \ldots, \mathsf{A}_n$. To compute a function f on a ciphertext:

- The client sends $\mathbf{e}^{(1)}$ to the first authority, who decrypts it to obtain $(\boldsymbol{x} - \boldsymbol{r}^{(1)}, \mathcal{H}.\mathsf{Enc}(\Omega_1, \boldsymbol{r}^{(1)}))$. The first authority then runs $eval_f$ to compute $\gamma_1 := \mathcal{H}.\mathsf{Enc}(\Omega_1, f(\boldsymbol{x}) - f(\boldsymbol{r}^{(1)}))$, and "peels off" one layer of encryption by computing $\mathsf{sk}_1^f := \mathsf{Dec}(\mathsf{sk}_1, \gamma_1)$ and sends sk_1^f to the client.
- The client contacts the second authority by sending $\mathbf{e}^{(2)}$ and sk_1^f. The second authority decrypts $\mathbf{e}^{(2)}$ to get $(\boldsymbol{r}^{(1)} - \boldsymbol{r}^{(2)}, \mathcal{H}.\mathsf{Enc}(\Omega_2, \boldsymbol{r}^{(2)}))$ and runs $eval_f$, getting $\mathcal{H}.\mathsf{Enc}(\Omega_2, f(\boldsymbol{r}^{(2)}) - f(\boldsymbol{r}^{(1)}))$. It homomorphically adds sk_1^f to it and peels off its layer of encryption, getting $\mathsf{sk}_2^f := \mathcal{H}.\mathsf{Enc}(\Omega_3, f(\boldsymbol{x}) - f(\boldsymbol{r}^{(2)}))$, which is sent back to the client.
- Following the same process, eventually, the client will receive $\mathsf{sk}_n^f := \mathcal{H}.\mathsf{Enc}(\mathsf{pk}_{\mathsf{c}}^{\mathcal{H}}, f(\boldsymbol{x}) - f(\boldsymbol{r}^{(n)}))$ and finishes the decryption as in the case of 1 authority, since the client knows $\boldsymbol{r}^{(n)}$.

Remark. Note that an advantage of our scheme is that the setup is not really a distributed protocol and authorities can obtain msk shares without interaction. On the other hand, notice that our construction imposes a specific order of communication with the authorities. It is an appealing target for future work to remove this limitation.

We will reduce the security of our multi-authority construction to the security of the construction for 1 authority (from Section 5.1). The following lemma is what makes the reduction possible. Roughly, it describes a method for perfectly simulating a ciphertext for $n+1$ authorities from a ciphertext for just n authorities. For lack of space the proof is in Appendix A.

Lemma 3. *Let $(r_n, \mathsf{e}^{(1)}, \ldots, \mathsf{e}^{(n)})$ be a ciphertext that honestly encrypts x for n authorities, and let $\mathsf{sk}_1^{\mathcal{E}}$ be the CCA2 secret key of the first authority. Let $(\mathsf{pk}_0^{\mathcal{H}}, \mathsf{sk}_0^{\mathcal{H}})$ and $(\mathsf{pk}_0^{\mathcal{E}}, \mathsf{sk}_0^{\mathcal{E}})$ be the homomorphic key pair and CCA2 key pair respectively of a new authority, and let λ_0 be an arbitrary policy. In these conditions, there is an efficient algorithm that, on input $(\mathsf{pk}_0^{\mathcal{H}}, \mathsf{sk}_0^{\mathcal{H}}, \mathsf{pk}_0^{\mathcal{E}}, \mathsf{sk}_1^{\mathcal{E}}, \mathsf{pk}_1^{\mathcal{E}}, r_n, \mathsf{e}^{(1)}, \ldots, \mathsf{e}^{(n)})$, can generate a ciphertext $(r_n, \widetilde{\mathsf{e}}^{(0)}, \widetilde{\mathsf{e}}^{(1)}, \mathsf{e}^{(2)}, \ldots, \mathsf{e}^{(n)})$ identically distributed to an honest encryption of x for $n+1$ authorities, in which the new authority is in the first position, followed by the original authorities (in the same order).*

Theorem 4 (Hbc authorities)**.** *Let $\mathcal{H}$ be a linearly-homomorphic encryption scheme and let $\mathcal{E}$ be a public-key encryption scheme. Then, the scheme from Figure 4 is such that for every adversary $\mathcal{A}$ in the* real world *(passively corrupting the n authorities), there exists a simulator $\mathcal{S}$ in the* ideal world *(corrupting the n authorities) who produces an output that is indistinguishable from the output of $\mathcal{A}$.*

Proof. We proceed by induction on the number of authorities. For $n = 1$, the result is true as established by Theorem 2. Now, assume the result is true for n authorities and consider the case of $n+1$ authorities. We pro-

ceed by *reductio ad absurdum*: assume that there exists a p.p.t. adversary $\mathcal{A}$ (passively controlling $n{+}1$ authorities) that produces a view which is distinguishable from the view that any simulator can generate in the ideal world. We could use this adversary to create a similar adversary $\mathcal{A}'$ (in a game of n authorities) that cannot be simulated (which contradicts the induction hypothesis).

Algorithm $\mathcal{A}'$ starts its experiment (in the real world of n authorities) and runs $\mathcal{E}.\mathsf{KeyGen}(1^\kappa)$ and $\mathcal{H}.\mathsf{KeyGen}(1^\kappa)$, one extra time (for the new simulated authority), producing $(\mathsf{pk}_0^{\mathcal{E}}, \mathsf{sk}_0^{\mathcal{E}})$ and $(\mathsf{pk}_0^{\mathcal{H}}, \mathsf{sk}_0^{\mathcal{H}})$. It runs the code of $\mathcal{A}$ with the exception that, when an evaluation request is performed, say on $(f, \mathsf{e}_1, \ldots, \mathsf{e}_\ell)$, for every $j \in [\ell]$ it modifies every vector e_j as described in Lemma 3 (for every $j \in [\ell]$) and continues executing the code of $\mathcal{A}$ with the simulated ciphertext for one extra authority. Recall that $\mathcal{A}'$ controls all the n authorities and thus knows all their secret keys, and this enables $\mathcal{A}'$ to run the algorithm of Lemma 3. Observe that $\mathcal{A}'$ is perfectly simulating the experiment executed by $\mathcal{A}$ (with $n{+}1$ authorities), and so, no simulator can exist for $\mathcal{A}'$ in the ideal world: *absurdum*. □

Theorem 5 (Malicious clients and hbc authorities)**.** *Let $\mathcal{H}$ be a linearly-homomorphic encryption scheme and let $\mathcal{E}$ be a CCA2 secure public-key encryption scheme. Then, the scheme from Figure 4 is such that for every adversary $\mathcal{A}$ in the* real world *(actively corrupting a subset of clients and passively corrupting up to $n{-}1$ authorities), there exists a simulator $\mathcal{S}$ in the* ideal world *(corrupting the same parties) who produces an output that is indistinguishable from the output of $\mathcal{A}$.*

Proof. Again, by induction on n, we have that for $n = 1$ the result is immediately implied by Theorem 3. Now, assume the result is true for n authorities and consider the case of $n{+}1$ authorities. Assume that there exists a p.p.t. adversary $\mathcal{A}$ (passively controlling a subset S of the $n{+}1$ authorities such that $|S| \leq n$, and actively corrupting a subset of clients) that produces a view which is distinguishable from the view that any simulator can generate in the ideal world. First, we can easily transform such an adversary into one that controls exactly n out of $n{+}1$ authorities (by acting genuinely in the name of those that were not controlled before). Therefore, from now on we assume that $\mathcal{A}$ controls exactly n authorities. As in the previous proof, we build an adversary $\mathcal{A}'$ that runs in the game with n authorities, and controls $n-1$ of them. We show that such an $\mathcal{A}'$ can simulate the behaviour of $\mathcal{A}$, and therefore, cannot be simulated. We denote with indices $\{1, \ldots, n\}$ the n authorities in the game of $\mathcal{A}'$. We distinguish two mutually exclusive cases regarding the authorities corrupted by $\mathcal{A}$:

1. *The last authority (in the game with $n{+}1$) is honest.* In this case, the first two authorities in the game with $n{+}1$ must be under the control of the adversary. Let us use indices $0, \ldots, n$ to refer to these $n{+}1$ authorities. In this case, $\mathcal{A}'$ proceeds as in Theorem 4. Namely, $\mathcal{A}'$ runs in a game with authorities $\{1, \ldots, n\}$, it samples the keys for the authority 0, $(\mathsf{pk}_0^{\mathcal{E}}, \mathsf{sk}_0^{\mathcal{E}})$, $(\mathsf{pk}_0^{\mathcal{H}}, \mathsf{sk}_0^{\mathcal{H}})$, and uses these keys, as well as $\mathsf{sk}_1^{\mathcal{E}}$, to run the algorithm of Lemma 3 in order to modify a ciphertext for authorities $\{1, \ldots, n\}$ into one for authorities $\{0, \ldots, n\}$ to be given to $\mathcal{A}$.
2. *The last authority (in the game with $n{+}1$) is corrupted.* In this case, the ciphertext expansion is easier as it does not require knowledge of any secret key. Let us use indices $\{1, \ldots, n+1\}$ to refer to the $n{+}1$ authorities in the game of $\mathcal{A}$, and let $\{1, \ldots, n\}$ the ones in the game of $\mathcal{A}'$. When a corrupted client (with public key $\mathsf{pk}_{\mathsf{c}}^{\mathcal{H}}$) receives a ciphertext, say $(r_n, (\mathsf{e}^{(1)}, \ldots, \mathsf{e}^{(n)}))$, $\mathcal{A}'$ generates a new fresh randomness $w \xleftarrow{\$} \mathcal{M}$ and sets

$$\mathsf{e}^{(n+1)} := \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}_{n+1}^{\mathcal{E}}}(r_n - w, \mathcal{H}.\mathsf{Enc}(\{\mathsf{pk}_{\mathsf{c}}^{\mathcal{H}}, \mathsf{pk}_{n+1}^{\mathcal{H}}\}, w), \lambda)$$

   where λ is chosen as $\mathcal{A}$ dictates. It simulates the $n{+}1$-authorities ciphertext as $(w, (\mathsf{e}^{(1)}, \ldots, \mathsf{e}^{(n)}, \mathsf{e}^{(n+1)}))$.

Observe that (in both cases) $\mathcal{A}'$ perfectly simulates the experiment executed by $\mathcal{A}$ (with $n{+}1$ authorities), therefore, no simulator can exist for $\mathcal{A}'$ in the ideal world.

Finally, we also observe that the need of distinguish between the two cases comes from the fact that we need to guarantee that $\mathcal{A}'$ controls strictly less than n authorities, and thus the authority that is kept honest by $\mathcal{A}$ must be present in the game run by $\mathcal{A}'$. □

Remark. Our mC-FE construction requires a PKI for clients, because data-owners must use a specific client's public key when encrypting their data. Note that the original C-FE construction of [37] also requires authenticated channels towards clients, albeit authentication in [37] does not necessarily need a PKI. An alternative to forgo the PKI for clients in our construction could be achieved if data-owners created a fresh key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{H}.\mathsf{KeyGen}$ at the moment of encryption and used pk as the client's public key, including both pk and sk in the ciphertext.

This alternative may compromise the *multi-input* property: if a single data-owner encrypts several ciphertexts, it can include the same pair of keys in all of them,

```
(* Mean *)

sum := 0;

for i in [m]{
  sum += x[i];
}
return sum.
```

```
(* Variance *)

sum1 := 0;
sum2 := 0;

for i in [m]{
  sum1 += x[i];
  sum2 += x[i]**2;
}
return m*sum2 -sum1**2.
```

Fig. 5. Example of input files in our programming language. Symbol `x` is reserved to represent the input vector and `m` is reserved to represent its length.

however, the keys will not match on ciphertexts produced by different data-owners. Yet, if the LHE scheme is *multi-key*, the client could combine the different public keys generated by different users into a single one. This would allow multi-input computations even if client's public keys are generated by different data-owners.

6 Implementation and Evaluation

We implement a general library for controlled functional encryption, based on our constructions. Our library is written in OCaml and uses the Relic-Toolkit [7] for elliptic curves. All the experiments were executed on an Intel Core i7-4790K CPU and 16GB of RAM, running Ubuntu 16.04 LTS.

We implement the CCA2 public-key encryption as a hybrid encryption scheme that uses AES (128-bits key) and the Cramer Shoup PKE scheme [23], implemented over the NIST P224 Curve. For the sake of simplicity, we ignore policies λ.

We have designed a simple programming language to describe circuits, which gives users fine-grained control on the way functions are evaluated. As an example, in Figure 5, we describe an implementation of the *mean* and *variance* functions (used in our experiments of linear regression in Section 6.3).

Observe that, as is common, we implement the mean multiplied by m (i.e. the sum) instead of the mean (both functions are equivalent if m is known) and we implement the variance multiplied by m^2 instead of the actual variance. This is to ensure that the result is an integer, a requirement imposed by our representation of numbers in discrete modular arithmetic.

6.1 Description of Our Experiments

In Section 6.2, we show how our construction can be used to evaluate quadratic classifiers over encrypted data. As in [42], we consider the popular MNIST dataset [33], a database of images of handwritten digits, commonly used to assess the performance of a machine learning classifier. It contains a total of 60 000 training images and 10 000 test images, where every image has been classified and labeled with a digit from 0 to 9. An image is represented as a vector of 784 (28×28) integers in the range $[0, 255]$, corresponding to the luminosity of every pixel.

The authors of [42], train a quadratic classifier that predicts the digit associated to an encrypted image with an accuracy of 97.54% on the test set.[4] More precisely, they use the training set to find a matrix P (of dimension $k \times 784$) and diagonal matrices D_i (of dimension $k \times k$) with integer coefficients, for every $i \in [0, 9]$. Now, an image $\boldsymbol{x} \in \mathbb{Z}^{784}$ is classified as digit j if $f_j(\boldsymbol{x}) = \max_{i \in [0,9]} \{f_i(\boldsymbol{x})\}$ where $f_i(\boldsymbol{x})$ is defined as $(P\boldsymbol{x})^\top D_i (P\boldsymbol{x})$ for every i. In Section 6.2 we use the exact same quadratic classifier from [42] (where $k = 40$) to evaluate it on randomly chosen images from the test set with our C-FE construction (see Table 1).

In case both data and classifier must be private we can leverage the multi-input property of our construction (for linear classifiers). More concretely, a linear classifier can be defined as a collection of pairs $(v_i, \boldsymbol{w}_i) \in \mathbb{Z} \times \mathbb{Z}^{784}$, for i in the set of classes Ψ; such that an image $\boldsymbol{x}$ is classified as belonging to class j if $g_j(\boldsymbol{x}) = \max_{i \in \Psi} \{g_i(\boldsymbol{x})\}$, where $g_i(\boldsymbol{x}) := v_i + \boldsymbol{w}_i^\top \boldsymbol{x}$ for every $i \in \Psi$. Such a classification can be privately performed with our C-FE construction if one party encrypts $(v_i, \boldsymbol{w}_i)$ for every $i \in \Psi$, and another party encrypts an image $\boldsymbol{x}$. Then, an evaluator (with the help of one or more authorities) computes the corresponding quadratic functions on both sources of encrypted data to obtain the values of $g_j(\boldsymbol{x})$. For our experiments (see Table 2), we consider the classifier that, given a specific digit, classifies an image in two classes, depending on whether it is the image of the given digit or not. Our ten different linear classifiers (one for each digit) trained with TensorFlow [1], achieve an accuracy on the test set between 96.14% and 99.39%.

4 Classifying encrypted images of hand-written digits allows to determine the digit itself while keeping any other information private. For example, the classification outcome does not leak the handwriting style, so to keep the authorship private.

	Encryption (s)	KeyReq (s)	Dec (ms)
ElGamal	0.61 ± 0.11	2.50 ± 0.42	2450 ± 580
Paillier	27.7 ± 2.5	6.10 ± 0.23	430 ± 36

Table 1. Evaluation times of the quadratic classifier.

	Encryption (s)			
	Image	Classifier	KeyReq (s)	Dec (ms)
ElGamal	0.58 ± 0.03	1.18 ± 0.05	2.94 ± 0.14	26 ± 6
Paillier	26.8 ± 3.1	53.5 ± 5.7	20.6 ± 1.6	78 ± 7

Table 2. Evaluation times of the linear classifier.

Finally, in Section 6.3 we show the performance of our multi-authority C-FE construction, used to performed linear regression over encrypted data using a small dataset of blood pressure measurements [44].

6.2 Single-Authority C-FE Experiments

Tables 1 and 2 show our results on applying the quadratic and the linear classifier, respectively, to randomly chosen encrypted images (from the test set). We compare the evaluation times, showing the average time of 20 evaluations (with a 95% confidence interval), between two different LHE scheme choices: ElGamal encryption [26], over the NIST P224 Curve, and Paillier cryptosystem [41], with a 2048-bits[5] modulus.

As expected, ElGamal performs much better, however Paillier has the advantage that the message space is exponentially large. When using ElGamal as a LHE scheme, decryption requires discrete logarithm computations. We have implemented the *baby-step giant-step algorithm* [38], with a precomputed table of 1.7GB that allows us to recover the discrete logarithm of 32-bits messages in roughly of 3ms. Exceptionally, the quadratic classifier of [42], produces 37-bits values—with the same storage we recover the message in roughly 0.2s. The encryption times in both tables (in seconds) represent the time of encrypting one image, i.e., 28×28 values. In the case of the linear classifier (Table 2), we additionally include the time of encrypting the classifier (a classifier consists of 785×2 values). Finally, note that decryption times are expressed in milliseconds.

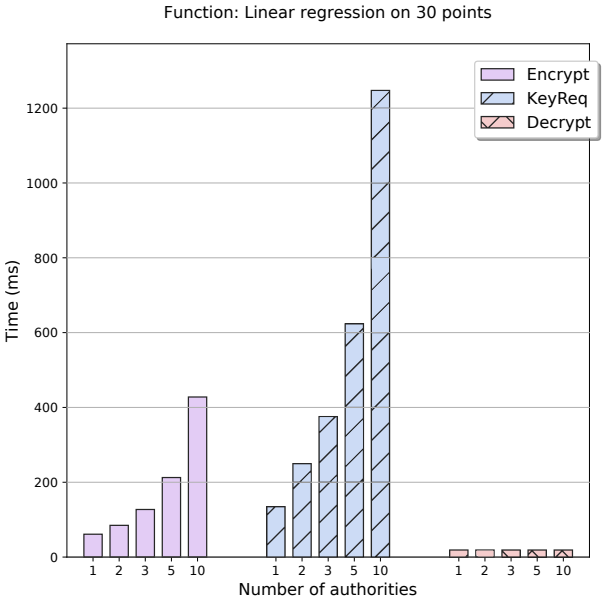

Fig. 6. Evaluation of our n-authority mC-FE.

	Paillier	ElGamal				
Authorities	1	1	2	3	5	10
Plaintext	256B	4B	4B	4B	4B	4B
Ciphertexts	768B	112B	196B	280B	448B	868B
mC-FE Keys	512B	56B	112B	168B	280B	560B

Table 3. Sizes (plaintexts, ciphertexts and keys) of our mC-FE

6.3 Multi-Authority C-FE Experiments

Since our main construction of mC-FE requires the LHE scheme have the distributed threshold property, experiments in this section are based on ElGamal.

As a proof of concept example, we consider a small medical record with data of 30 patients [44]. We encrypt the age and systolic blood pressure of every patient and evaluate simple linear regression on the encrypted data. Figure 6 shows the median times (of 20 executions) for encryption, key request and decryption, where four statistics are computed in order to derive the regression line: average age, average blood pressure, variance of the age variable and covariance of both variables. In the case of key request, we aggregate the execution time of all authorities. As expected, the number of authorities slows down encryption and key request linearly. On the other hand, decryption times are independent on the number of authorities.

5 To achieve a comparable security level of 112-bits.

Finally, in Table 3 we show the sizes relative to the implementation of our construction. Entries *plaintext* and *ciphertext* correspond to the size of encrypting one single integer[6], e.g., encrypting an MNIST image (which is encoded as 784 integers) with ElGamal and 3 authorities would require about $280\text{B} \cdot 784 = 214\text{KB}$. Note that the size of mC-FE keys is independent of the input length and the underlying function.

7 Conclusions

Controlled Functional Encryption (C-FE) enables clients to mine on third-party encrypted data while an authority exerts fine-grained access control on *who* can mine the data, *what* information can be extracted, and *which* ciphertexts can be used for mining. Thus, C-FE emerges as a promising primitive to balance utility and privacy in data mining applications that deal with sensitive data. Previous work has proposed a C-FE for single-input linear functions based on CCA2 encryption and one for arbitrary functions based on garbled circuits.

In this paper we have revised the notion of C-FE and extended it in multiple directions. In particular, we have defined multi-authority C-FE (or mC-FE) as a C-FE scenario where access control is carried out by a cohort of authorities. We have then provided an instantiation for quadratic functions (on inputs potentially coming from multiple data-owners) that leverages CCA2 encryption and linear homomorphic encryption (thereby forgoing garbled circuits). We have also implemented and evaluated our proposal.

A problem left open by our instantiations is to (efficiently) tolerate malicious authorities. Indeed, both our instantiations (the single-authority and the multi-authority) are proven secure against malicious clients but hbc authorities. While the support for malicious clients is rather straightforward since in our protocol the client is "almost" passive[7], supporting malicious authorities would require a mechanism for the client to check that each authority has performed the computation correctly. Concretely, for our constructions, this could be achieved by designing an efficient verifiable computation mechanism for the degree-2 homomorphic computation based on LHE. We leave the investigation of this extension as future work.

6 The size of the integer is bounded by the message space, i.e., 224-bits in the case of ElGamal or 2048-bits with Paillier. However, in the case of ElGamal, decryption may be inefficient unless small integers (for example, 4-bytes integers) are used.

7 The only active attack it can mount is to alter the information passed to the authorities which is not possible thanks to the CCA2 security of the encryption scheme.

Acknowledgments

The research of Dario Fiore has been partially supported by the Spanish Government under projects SCUM (ref. RTI2018-102043-B-I00), CRYPTOEPIC (refs. ERC2018-092822, EUR2019-103816), and SECURITAS (ref. RED2018-102321-T), and by the Madrid Regional Government under project BLOQUES (ref. S2018/TCS-4339), and by a research contract with NEC Laboratories Europe GmbH. Claudio Soriente has been partially funded by the EU H2020-SU-ICT-03-2018 Project No. 830929 CyberSec4Europe and by the European Union's Horizon 2020 research and innovation programme under grant agreement No 779852.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] M. Abdalla, F. Benhamouda, M. Kohlweiss, and H. Waldner. Decentralizing inner-product functional encryption. In *Public Key Cryptography (2)*, volume 11443 of *Lecture Notes in Computer Science*, pages 128–157. Springer, 2019.

[3] M. Abdalla, F. Bourse, A. De Caro, and D. Pointcheval. Simple functional encryption schemes for inner products. In *PKC 2015*, LNCS, pages 733–751. Springer, 2015.

[4] M. Abdalla, D. Catalano, D. Fiore, R. Gay, and B. Ursu. Multi-input functional encryption for inner products: Function-hiding realizations and constructions without pairings. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, pages 597–627, 2018.

[5] M. Abdalla, R. Gay, M. Raykova, and H. Wee. Multi-input inner-product functional encryption from pairings. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*, pages 601–626, 2017.

[6] S. Agrawal, B. Libert, and D. Stehlé. Fully secure functional encryption for inner products, from standard assumptions.

LNCS, pages 333–362. Springer, Aug. 2016.

[7] D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic.

[8] S. Badrinarayanan, D. Gupta, A. Jain, and A. Sahai. Multi-input functional encryption for unbounded arity functions. In *Advances in Cryptology – ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 27–51. Springer, Dec. 2015.

[9] C. E. Z. Baltico, D. Catalano, D. Fiore, and R. Gay. Practical functional encryption for quadratic functions with applications to predicate encryption. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, pages 67–98, 2017.

[10] M. Barbosa, D. Catalano, and D. Fiore. Labeled homomorphic encryption. In S. N. Foley, D. Gollmann, and E. Snekkenes, editors, *Computer Security – ESORICS 2017*, pages 146–166, Cham, 2017. Springer International Publishing.

[11] A. Bishop, A. Jain, and L. Kowalczyk. Function-hiding inner product encryption. In *Advances in Cryptology – ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 470–491. Springer, Dec. 2015.

[12] D. Boneh, A. Raghunathan, and G. Segev. Function-private identity-based encryption: Hiding the function in functional encryption. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 461–478. Springer, Aug. 2013.

[13] D. Boneh, A. Raghunathan, and G. Segev. Function-private subspace-membership encryption and its applications. In K. Sako and P. Sarkar, editors, *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 255–275. Springer, Dec. 2013.

[14] D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In Y. Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 253–273. Springer, Mar. 2011.

[15] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser. Machine learning classification over encrypted data. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*, 2015.

[16] E. Boyle, K.-M. Chung, and R. Pass. On extractability obfuscation. In Y. Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 52–73. Springer, Feb. 2014.

[17] Z. Brakerski and G. Segev. Function-private functional encryption in the private-key setting. In *TCC 2015: 12th Theory of Cryptography Conference, Part II*, volume 9015 of *LNCS*, pages 306–324. Springer, 2015.

[18] D. Catalano and D. Fiore. Using linearly-homomorphic encryption to evaluate degree-2 functions on encrypted data. In *ACM CCS 15*, pages 1518–1529. ACM Press, 2015.

[19] N. Chandran, V. Goyal, A. Jain, and A. Sahai. Functional encryption: Decentralised and delegatable. *IACR Cryptology ePrint Archive*, 2015:1017, 2015.

[20] J. Chotard, E. Dufour Sans, R. Gay, D. H. Phan, and D. Pointcheval. Decentralized multi-client functional encryption for inner product. In *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II*, pages 703–732, 2018.

[21] H. Corrigan-Gibbs and D. Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, pages 259–282, 2017.

[22] V. Cortier, D. Galindo, S. Glondu, and M. Izabachène. Distributed elgamal à la pedersen: Application to helios. In *Proceedings of the 12th annual ACM Workshop on Privacy in the Electronic Society, WPES 2013, Berlin, Germany, November 4, 2013*, pages 131–142, 2013.

[23] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In H. Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 13–25. Springer, Aug. 1998.

[24] P. Datta, R. Dutta, and S. Mukhopadhyay. Functional encryption for inner product with full function privacy. LNCS, pages 164–195. Springer, 2016.

[25] A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung. How to share a function securely. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '94, page 522–533, New York, NY, USA, 1994. Association for Computing Machinery.

[26] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and D. Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 10–18. Springer, Aug. 1984.

[27] B. Fisch, D. Vinayagamurthy, D. Boneh, and S. Gorbunov. IRON: functional encryption using intel SGX. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 765–782, 2017.

[28] 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, Oct. 2013.

[29] S. Garg, C. Gentry, S. Halevi, and M. Zhandry. Functional encryption without obfuscation. In E. Kushilevitz and T. Malkin, editors, *TCC 2016-A: 13th Theory of Cryptography Conference, Part II*, volume 9563 of *LNCS*, pages 480–511, Tel Aviv, Israel, Jan. 10–13, 2016. Springer.

[30] S. Goldwasser, S. D. Gordon, V. Goyal, A. Jain, J. Katz, F.-H. Liu, A. Sahai, E. Shi, and H.-S. Zhou. Multi-input functional encryption. In P. Q. Nguyen and E. Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 578–602. Springer, May 2014.

[31] T. Graepel, K. E. Lauter, and M. Naehrig. ML confidential: Machine learning on encrypted data. In *Information Security and Cryptology - ICISC 2012 - 15th International Conference, Seoul, Korea, November 28-30, 2012, Revised Selected Papers*, pages 1–21, 2012.

[32] H. Kilinc and A. Küpçü. Optimally efficient multi-party fair exchange and fair secure multi-party computation. 04 2015.

[33] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010.

[34] H. Lin and S. Tessaro. Indistinguishability obfuscation from trilinear maps and block-wise local PRGs. LNCS, pages 630–660. Springer, 2017.

[35] L. Melis, G. Danezis, and E. D. Cristofaro. Efficient private statistics with succinct sketches. In *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San*

Diego, California, USA, February 21-24, 2016, 2016.

[36] P. Mohassel and Y. Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 19–38, 2017.

[37] M. Naveed, S. Agrawal, M. Prabhakaran, X. Wang, E. Ayday, J.-P. Hubaux, and C. A. Gunter. Controlled functional encryption. In G.-J. Ahn, M. Yung, and N. Li, editors, *ACM CCS 14*, pages 1280–1291. ACM Press, Nov. 2014.

[38] V. I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55(2):165–172, Feb 1994.

[39] V. Nikolaenko, S. Ioannidis, U. Weinsberg, M. Joye, N. Taft, and D. Boneh. Privacy-preserving matrix factorization. In A.-R. Sadeghi, V. D. Gligor, and M. Yung, editors, *ACM CCS 13*, pages 801–812. ACM Press, Nov. 2013.

[40] A. O'Neill. Definitional issues in functional encryption. *IACR Cryptology ePrint Archive*, 2010:556, 2010.

[41] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In J. Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, May 1999.

[42] E. D. Sans, R. Gay, and D. Pointcheval. Reading in the dark: Classifying encrypted digits with functional encryption. Cryptology ePrint Archive, Report 2018/206, 2018. https://eprint.iacr.org/2018/206.

[43] E. Shen, E. Shi, and B. Waters. Predicate privacy in encryption systems. In O. Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 457–473. Springer, Mar. 2009.

[44] H. Spaeth. Mathematical algorithms for linear regression. Academic Press, page 304, 1991.

[45] B. Waters. A punctured programming approach to adaptively secure functional encryption. In *CRYPTO 2015, Part II*, LNCS, pages 678–697. Springer, Aug. 2015.

A Proofs

Proof of Theorem 2.

We show that for every adversary $\mathcal{A}$ (passively controlling the authority), there exists a simulator $\mathcal{S}$ such that no p.p.t. environment $\mathcal{D}$ can distinguish between the view of $\mathcal{A}$ in the *real world* and the simulated view by $\mathcal{S}$ in the *ideal world*. For that, we define a sequence of games, starting from the *real world experiment* and progressively modifying it until we reach a last game, that we can simulate in the *ideal world*. All games consist of an execution of a C-FE experiment, and the output of every game is defined as the final view of the corrupted parties at the end of the experiment.

- **Game $\mathbf{G_0}$**: The real world experiment.
- **Game $\mathbf{G_{1.i}}$**: The real world experiment, with the exception that the first i ciphertexts sent during evaluation requests are slightly modified: for data x and policy λ, instead of being created as $r \xleftarrow{\$} \mathcal{M}$, $\mathcal{E}.\mathsf{Enc}(\mathsf{mpk}, (x-r, \mathcal{H}.\mathsf{Enc}(\mathsf{pk}_\mathsf{c}^\mathcal{H}, r), \lambda))$, they are created as $\mathcal{E}.\mathsf{Enc}(\mathsf{mpk}, (y, \mathcal{H}.\mathsf{Enc}(\mathsf{pk}_\mathsf{c}^\mathcal{H}, r), \lambda))$ for $y, r \xleftarrow{\$} \mathcal{M}$.
- **Game $\mathbf{G_2}$**: Game G_1, except that *all* the ciphertexts sent for evaluation requests are modified as above.

Let q be an upper-bound on the number of ciphertexts sent by $\mathcal{A}$ to the authority on evaluation requests. We first argue that any p.p.t. distinguisher between G_0 and G_2 has a distinguishing advantage upper-bounded by $q \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{H},\mathcal{B}}(\kappa)$, for some p.p.t. algorithm $\mathcal{B}$. Note that G_0 equals $G_{1.0}$ and that $G_{1.q}$ equals G_2. Therefore, it is enough to prove that the distinguishing advantage of any external environment between $G_{1.i}$ and $G_{1.(i+1)}$ (for a generic i) is upper-bounded by $\mathsf{Adv}^{\mathsf{Msk}\mathit{vs}\mathsf{Rnd}}_{\mathcal{H},\mathcal{B}}(\kappa)$ (and then apply Lemma 1 and the union bound).

Now, let $\mathcal{D}$ be a p.p.t. distinguisher between the games $G_{1.i}$ and $G_{1.(i+1)}$. We build a p.p.t. algorithm $\mathcal{B}$ against the Msk-vs-Rnd experiment on $\mathcal{H}$ (Lemma 1), that succeeds with at least the same probability $\mathcal{D}$ does. The challenger of $\mathcal{B}$ will run $(\mathsf{pk}_\mathsf{c}^\mathcal{H}, \mathsf{sk}_\mathsf{c}^\mathcal{H}) \leftarrow \mathcal{H}.\mathsf{KeyGen}(1^\kappa)$ and will send $\mathsf{pk}_\mathsf{c}^\mathcal{H}$ to $\mathcal{B}$. Now, $\mathcal{B}$ starts running game $G_{1.i}$ (which so far is identical to game $G_{1.(i+1)}$). For that, it executes $\Upsilon.\mathsf{Setup}(1^\kappa)$ to get a pair of keys $(\mathsf{mpk}, \mathsf{msk})$. It then assigns the public key $\mathsf{pk}_\mathsf{c}^\mathcal{H}$ to the client that will perform the evaluation request involving the i+1-th ciphertext. For the rest of evaluation requests, it samples a pair of client keys with $\mathcal{H}.\mathsf{KeyGen}$ when a new identity interacts with the authority and feeds the authority as the environment $\mathcal{D}$ dictates. Algorithm $\mathcal{B}$ simulates the evaluation requests as the authority would do (using its own msk). Eventually, an evaluation request will contain the i+1-th ciphertext. Let this evaluation request be $(f, \mathsf{e}_1, \ldots, \mathsf{e}_\ell)$, and let e_k, for some $k \in [\ell]$ be the i+1-th ciphertext during the whole execution. Because e_k was created by an honest data-owner (the only corrupted party is the authority), $\mathcal{B}$ has created it by executing $\mathcal{E}.\mathsf{Enc}(\mathsf{mpk}, (a, b, \lambda))$, where r was sampled $r \xleftarrow{\$} \mathcal{M}$, a was set to $x-r$, and b was set to $\mathcal{H}.\mathsf{Enc}(\mathsf{pk}_\mathsf{c}^\mathcal{H}, r)$. At this point, $\mathcal{B}$ starts the communication (of the Msk-vs-Rnd game) with its challenger and sends its first message x. The challenger will respond with (y_{b^*}, ct), where $\mathsf{ct} \coloneqq \mathsf{Enc}(\mathsf{pk}, r)$ of some $r \xleftarrow{\$} \mathcal{M}$ and y_{b^*} is either $x-r$ (if $b^* = 1$) or uniformly random from $\mathcal{M}$ (if $b^* = 0$). Now, $\mathcal{B}$ replaces this ciphertext and continues the execution of the game, sending $(f, \mathsf{e}_1, \ldots, \mathsf{e}_{k-1}, (y, \mathsf{ct}), \mathsf{e}_{k+1}, \ldots, \mathsf{e}_\ell)$, and it completes the execution of the game (either $G_{1.i}$ or $G_{1.(i+1)}$, because they are equivalent from this point).

After finishing the execution, it sends the produced view to $\mathcal{D}$, who will output a bit. Algorithm $\mathcal{B}$ passes this bit to its challenger. Simply, observe that if $b^* = 1$, $\mathcal{B}$ has just perfectly simulated an execution of $G_{1.i}$, while if $b^* = 0$, $\mathcal{B}$ has just perfectly simulated an execution of $G_{1.(i+1)}$. Therefore, if $\mathcal{D}$ succeeds in differentiating between these games, $\mathcal{B}$ will win the Msk-vs-Rnd game, as desired.

Finally, we provide a simulator $\mathcal{S}$ that perfectly simulates G_2 in the *ideal world*. Simulator $\mathcal{S}$ first runs $\Upsilon.\mathsf{Setup}(1^\kappa)$ to get a pair of keys $(\mathsf{mpk}, \mathsf{msk})$, and adds both keys to the simulated view.

Since the adversary only controls the authority, we just need to show how to simulate evaluation requests. The simulator initializes a table T^h of handles and a table T^{id} of identities. Observe that, in the ideal world, on every evaluation request, $\mathcal{O}$ sends $(t, f, \mathsf{C}, \boldsymbol{h}, \boldsymbol{\lambda}, 1)$ to the simulator. On input this information, the simulator looks in its table T^{id} for $(\mathsf{C}, \mathsf{pk}_\mathsf{C}^\mathcal{H})$. If this entry does not exist, it samples $(\mathsf{pk}_\mathsf{C}^\mathcal{H}, \mathsf{sk}_\mathsf{C}^\mathcal{H}) \leftarrow \mathcal{H}.\mathsf{KeyGen}(1^\kappa)$ and adds $(\mathsf{C}, \mathsf{pk}_\mathsf{C}^\mathcal{H})$ to T^{id}. For every handle in $\boldsymbol{h}$, h_j, $\mathcal{S}$ looks in T^h for the entry (h_j, e_j). If it does not exist, it samples $y, r \xleftarrow{\$} \mathcal{M}$ and sets $\mathsf{e}_j \coloneqq \mathcal{E}.\mathsf{Enc}(\mathsf{mpk}, (y, \mathcal{H}.\mathsf{Enc}(\mathsf{pk}_\mathsf{C}^\mathcal{H}, r), \lambda_j))$ and adds the pair (h_j, e_j) to T^h. Finally, it adds $(f, \mathsf{e}, \boldsymbol{\lambda})$ and $\mathsf{pk}_\mathsf{C}^\mathcal{H}$ to the view.

Note that $\mathcal{S}$ perfectly simulates the real world experiment of game G_2. □

Proof of Theorem 3.

We show that for every $t \in \mathbb{N}$ and for every adversary $\mathcal{A}$ (controlling t clients), there exists a simulator $\mathcal{S}$ such that no p.p.t. environment $\mathcal{D}$ can distinguish between the view of $\mathcal{A}$ in the *real world* and the simulated view by $\mathcal{S}$ in the *ideal world*. For that, we define a sequence of games, starting from the *real world experiment* and progressively modifying it until we reach a last game, that we can simulate in the *ideal world*. All games consist of an execution of a C-FE experiment, and the output of every game is defined as the final view of the corrupted parties at the end of the experiment.

- **Game $\mathbf{G_0}$**: The real world experiment.
- **Game $\mathbf{G_1}$**: The real world experiment, with the exception that every evaluation request performed by a malicious client, say $(f, \mathsf{e}_1, \ldots, \mathsf{e}_\ell)$, is handled by the authority in a slightly different way: on decrypting e_j (for every $j \in [\ell]$), if it corresponds to a ciphertext generated by an honest client, decryption is not executed; instead, the plaintext encrypted by the honest client is used.
- **Game $\mathbf{G_{2.i}}$**: Game G_1, except that the first i data uploads performed from an honest data-owner to a corrupted client, are modified: the ciphertext that the client receives, $(r, \mathbf{e})$, is replaced by $(r, \widetilde{\mathbf{e}})$, where[8] $\widetilde{\mathbf{e}} \coloneqq \mathcal{E}.\mathsf{Enc}(\mathsf{mpk}, \mathsf{str})$.
- **Game $\mathbf{G_3}$**: As G_1, except that *all* data uploads from honest data-owners to corrupted clients are modified as above.

The difference between G_0 and G_1 is imperceptible for $\mathcal{D}$, as the view of $\mathcal{A}$ is identical in both games.

Let u be an upper-bound on the number of data uploads performed by honest data-owners. We now argue that any p.p.t. distinguisher between G_1 and G_3 has a distinguishing advantage upper-bounded by $u \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CCA2}}_{\mathcal{E},\mathcal{B}}(\kappa)$, for some p.p.t. algorithm $\mathcal{B}$. Note that G_1 equals $G_{2.0}$ and that $G_{2.u}$ equals G_3. Therefore, it is enough to prove that the distinguishing advantage of any external environment between $G_{2.i}$ and $G_{2.(i+1)}$ (for a generic i) is upper-bounded by $\mathsf{Adv}^{\mathsf{IND\text{-}CCA2}}_{\mathcal{E},\mathcal{B}}(\kappa)$ (and then apply the union bound). Let $\mathcal{D}$ be a p.p.t. distinguisher between the games $G_{2.i}$ and $G_{2.(i+1)}$. We build a p.p.t. algorithm $\mathcal{B}$ against the IND-CCA2 security of $\mathcal{E}$ that succeeds with at least the same probability $\mathcal{D}$ does:

The challenger of $\mathcal{B}$ will run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{E}.\mathsf{KeyGen}(1^\kappa)$ and will send pk to $\mathcal{B}$. Now, $\mathcal{B}$ starts running game $G_{2.i}$ (which so far is identical to game $G_{2.(i+1)}$), on public key $\mathsf{mpk} \coloneqq \mathsf{pk}$ (identically distributed to a mpk generated from $\Upsilon.\mathsf{Setup}$), following the code of $\mathcal{A}$. For that, $\mathcal{B}$ creates t public keys for the t clients in the same way[9] $\mathcal{A}$ does.

On the j-th data upload, for every $j \in [i]$, (say, to the k-th corrupted client) on data[10] (x_j, λ_j): $\mathcal{B}$ samples $r_j \xleftarrow{\$} \mathcal{M}$, producing the ciphertext (r_j, e_j), with $\mathsf{e}_j \coloneqq \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, \mathsf{str})$; it also sets variable $\mathsf{ckey}_j = \mathsf{pk}_{\mathsf{C}\,k}^\mathcal{H}$.

On every evaluation request performed by the code of $\mathcal{A}$, it simulates the authority:

- if some of the input ciphertexts is one of the simulated for an honest data upload, i.e., it equals e_j, for some $j \in [i]$, $\mathcal{B}$ performs step (3) of KeyReq for that ciphertext as $a \coloneqq x_j - r_j$, $b \coloneqq \mathcal{H}.\mathsf{Enc}(\mathsf{ckey}_j, r_j)$, $\lambda = \lambda_j$,

8 Let str be an arbitrary string (for example, of zeros) whose length is the length of the elements of $\mathcal{M} \times \mathcal{C} \times \Lambda$.

9 It may not always be by running the official algorithm $\mathcal{H}.\mathsf{KeyGen}$, this depends on the adversary $\mathcal{A}$.

10 Note that without loss of generality, we can assume that $\mathcal{B}$ knows the data chosen by honest users in this execution, because $\mathcal{B}$ is quantified after the environment $\mathcal{D}$.

- otherwise, $\mathcal{B}$ performs step (3) of KeyReq by using its $\mathcal{O}_1$ oracle (provided by its challenger in the IND-CCA2 game),

and continues executing KeyReq as described. At some point, when the $(i{+}1)$-th data upload comes, (say, on data x_{i+1}, policy λ_{i+1} and for the k-th client), $\mathcal{B}$ will proceed as follows: it will sample $r_{i+1} \xleftarrow{\$} \mathcal{M}$, and it will set

$$m_0 \coloneqq (x_{i+1}{-}r_{i+1}, \mathcal{H}.\mathsf{Enc}(\mathsf{pk}^{\mathcal{H}}_{\mathsf{c}\,k}, r_{i+1}), \lambda_{i+1})$$
$$m_1 \coloneqq \mathsf{str}\,.$$

It will send (m_0, m_1) to its challenger, getting back the target ciphertext $\mathsf{ct}^* \coloneqq \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, m_{b^*})$ for some $b^* \in \{0,1\}$. Lastly, it will set the ciphertext corresponding to this data upload to (r_{i+1}, ct^*).

From this point on, $\mathcal{B}$ continues executing game $G_{2.i}$ (or $G_{2.(i+1)}$, they are identical again), by following the same process described above for data uploads and evaluation requests (this time with its oracle $\mathcal{O}_2$). Note that $\mathcal{B}$ will not have to call $\mathcal{O}_2$ on the target ciphertext, because if ct^* is sent in an evaluation request, it will be handled as in the first bullet point above.

When $\mathcal{B}$ finishes the execution, it passes the final view to distinguisher $\mathcal{D}$, who will output a bit. Finally, algorithm $\mathcal{B}$ copies the answer of $\mathcal{D}$ and outputs the same bit to its challenger. Observe that if $b^* = 0$, $\mathcal{B}$ is perfectly simulating $G_{2.i}$, while if $b^* = 1$, $\mathcal{B}$ is perfectly simulating $G_{2.(i+1)}$. Therefore, if $\mathcal{D}$ succeeds in differentiating between these games, $\mathcal{B}$ will win the IND-CCA2 game, as desired.

We finish the proof of Theorem 3 by defining a simulator $\mathcal{S}$ that perfectly simulates G_3 in the *ideal world*. Simulator $\mathcal{S}$ first runs $\Upsilon.\mathsf{Setup}(1^\kappa)$ to get a pair of keys $(\widetilde{\mathsf{mpk}}, \widetilde{\mathsf{msk}})$, and adds $\widetilde{\mathsf{mpk}}$ to the simulated view (which is identically distributed to the master public-key that $\mathcal{A}$ gets in the real world). It also creates t public keys for the t corrupted clients in the same way $\mathcal{A}$ does in game G_3, say $(\widetilde{\mathsf{pk}^{\mathcal{H}}_{\mathsf{c}}}\,1, \widetilde{\mathsf{sk}^{\mathcal{H}}_{\mathsf{c}}}\,1), \ldots, (\widetilde{\mathsf{pk}^{\mathcal{H}}_{\mathsf{c}}}\,t, \widetilde{\mathsf{sk}^{\mathcal{H}}_{\mathsf{c}}}\,t)$.

On the j-th data upload, the ideal functionality $\mathcal{O}$ will send to $\mathcal{S}$ a handle h_j and the identity of the corrupted client for which the data was intended, say it was for the k-th corrupted client. The simulator, now sets $\mathsf{ckey}_j \coloneqq \mathsf{pk}^{\mathcal{H}}_{\mathsf{c}\,k}$ and samples $r_j \xleftarrow{\$} \mathcal{M}$, producing the ciphertext $(r, \mathcal{E}.\mathsf{Enc}(\mathsf{pk}, \mathsf{str}))$, as in game G_3.

On every evaluation request described in the code of $\mathcal{A}$, say $(f, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)$ in the name of the k-th corrupted[11] client the simulator simulates the authority. For that, $\mathcal{S}$ first fills a vector of handles $(\widehat{h}_1, \ldots, \widehat{h}_\ell)$ and another vector of messages $(\widehat{r}_1, \ldots, \widehat{r}_\ell)$, i.e., for all $i \in [\ell]$:

- if ct_i is equal to one of the ciphertexts simulated for data uploads for honest users, that is, $\mathsf{ct}_i = \mathsf{ct}_j$ for some j, it sets $\widehat{h}_i \coloneqq h_j$ (the handle received in the j-th data upload), and $\widehat{r}_i \coloneqq r_j$ (the random message generated in the j-th data upload),
- otherwise, it sets $(\widetilde{a}_i, \widetilde{b}_i, \widetilde{\lambda}_i) \coloneqq \mathcal{E}.\mathsf{Dec}(\widetilde{\mathsf{msk}}, \mathsf{ct}_i)$; if $\widetilde{a}_i \notin \mathcal{M}$ or $\widetilde{b}_i \notin \mathcal{C}$ or $\widetilde{\lambda}_i \notin \Lambda$, $\mathcal{S}$ aborts the simulation of the authority, with $\widetilde{\mathsf{sk}}^f \coloneqq \bot$; else, it computes

$$\widetilde{r}_i \coloneqq \mathcal{H}.\mathsf{Dec}(\widetilde{\mathsf{sk}^{\mathcal{H}}_{\mathsf{c}}}\,k, \widetilde{b}_i) \quad \text{and} \quad \widetilde{x}_i \coloneqq \widetilde{a}_i + \widetilde{r}_i$$

  and sends a data upload to the ideal functionality $\mathcal{O}$, directed to the k-th corrupted client and for data $(\widetilde{x}_i, \widetilde{\lambda}_i)$, immediately receiving a handle h from $\mathcal{O}$, finally, $\mathcal{S}$ sets $\widehat{h}_i \coloneqq h$ and $\widehat{r}_i \coloneqq \widetilde{r}_i$.

After completing the vectors (of handles and messages), $\mathcal{S}$ sends an evaluation request to $\mathcal{O}$ as the k-th corrupted client on input $(f, \widehat{h}_1, \ldots, \widehat{h}_\ell)$, getting back (after the corresponding decryption query) a value $\widetilde{y}$ or $\bot$. To conclude the simulation of the authority in this evaluation request, $\mathcal{S}$ outputs

$$\widetilde{\mathsf{sk}}^f \coloneqq \mathcal{H}.\mathsf{Sim}\,(1^\kappa, \mathsf{pk}^{\mathcal{H}}_{\mathsf{c}\,k}, \widetilde{y} - f(\widehat{r}_1, \ldots, \widehat{r}_\ell))\quad .$$

Note that $\widetilde{\mathsf{sk}}^f$ is simulated perfectly, since it is identically distributed to the answer of the authority in the real world (game G_3): it is a ciphertext (of $\mathcal{H}$) that, decrypted with $\mathsf{pk}^{\mathcal{H}}_{\mathsf{c}\,k}$, results in $f(\boldsymbol{x}){-}f(\boldsymbol{r})$, where $\boldsymbol{x}$ and $\boldsymbol{r}$ are the hidden vectors in the evaluation request $(f, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)$. Finally, observe that the *context hiding* of the function $eval_f$ makes the computation of $\widetilde{\mathsf{sk}}^f$ identically distributed to the one performed by the authority in G_3 (through the $eval_f$ procedure). Formally, we would use the simulator $\widetilde{\mathsf{Sim}}$ mentioned in the remark about *context hiding* that follows Lemma 2. □

Proof of Lemma 1.

Let $\mathcal{A}$ be a p.p.t. adversary against the Msk-vs-Rnd security game. We build from $\mathcal{A}$ a p.p.t. adversary $\mathcal{B}$ against the IND-CPA experiment, which succeeds with at least the same probability (actually, the same) that $\mathcal{A}$ does in experiment Msk-vs-Rnd. On input pk, algorithm $\mathcal{B}$ challenges $\mathcal{A}$ on the same public key, receiving a message $x \in \mathcal{M}$. It then samples $s, r \xleftarrow{\$} \mathcal{M}$ and sets $m_0 \coloneqq s$

11 We assume clients must authenticate to the authority in order to perform evaluation requests and that the authority answers with $\bot$ if the authentication fails (e.g. because they try to impersonate an honest user); furthermore, we assume that policies λ include information about the clients who should have access to the data.

Addition, $\diamond = \oplus$, and so $f(\boldsymbol{x}) = f_1(\boldsymbol{x}) + f_2(\boldsymbol{x})$:

- f_1, f_2 constant: $(c_1, \bot) \oplus (c_2, \bot) = (c_1 + c_2, \bot)$, *i)*
- f_1 const., f_2 linear: $(c_1, \bot) \oplus \big(f_2(\boldsymbol{x}) - f_2(\boldsymbol{r}),\ \mathsf{Enc}(f_2(\boldsymbol{r}))\big) = \big(f(\boldsymbol{x}) - f(\boldsymbol{r}),\ c_1 \oplus \mathsf{Enc}(f_2(\boldsymbol{r}))\big)$, *ii)*
- f_1 const., f_2 quad.: $(c_1, \bot) \oplus \big(\bot, \mathsf{Enc}(f_2(\boldsymbol{x}) - f_2(\boldsymbol{r}))\big) = \big(\bot, \mathsf{Enc}(f(\boldsymbol{x}) - f(\boldsymbol{r}))\big)$, *iii)*
- f_1, f_2 linear: $\big(f_1(\boldsymbol{x}) - f_1(\boldsymbol{r}),\ \mathsf{Enc}(f_1(\boldsymbol{r}))\big) \oplus \big(f_2(\boldsymbol{x}) - f_2(\boldsymbol{r}),\ \mathsf{Enc}(f_2(\boldsymbol{r}))\big)$
  $= \big((f_1 + f_2)(\boldsymbol{x}) - (f_1 + f_2)(\boldsymbol{r}), \mathsf{Enc}((f_1 + f_2)(\boldsymbol{r}))\big)$, *ii)*
- f_1 linear, f_2 quad.: $\big(f_1(\boldsymbol{x}) - f_1(\boldsymbol{r}),\ \mathsf{Enc}(f_1(\boldsymbol{r}))\big) \oplus \big(\bot, \mathsf{Enc}(f_2(\boldsymbol{x}) - f_2(\boldsymbol{r}))\big)$
  $= (\bot, (\{f_1(\boldsymbol{x}) - f_1(\boldsymbol{r})\} \oplus \mathsf{Enc}(f_2(\boldsymbol{x}) - f_2(\boldsymbol{r}))))$, *iii)*
- f_1, f_2 quad.: $\big(\bot, \mathsf{Enc}(f_1(\boldsymbol{x}) - f_1(\boldsymbol{r}))\big) \oplus \big(\bot, \mathsf{Enc}(f_2(\boldsymbol{x}) - f_2(\boldsymbol{r}))\big) = \big(\bot, \mathsf{Enc}((f_1{+}f_2)(\boldsymbol{x}){-}(f_1{+}f_2)(\boldsymbol{r}))\big)$, *iii)*

Multiplication, $\diamond = \otimes$, and so $f(\boldsymbol{x}) = f_1(\boldsymbol{x})f_2(\boldsymbol{x})$:

- f_1, f_2 constant: $(c_1, \bot) \otimes (c_2, \bot) = (c_1c_2, \bot)$, *i)*
- f_1 const., f_2 linear: $(c_1, \bot) \otimes \big(f_2(\boldsymbol{x}) - f_2(\boldsymbol{r}),\ \mathsf{Enc}(f_2(\boldsymbol{r}))\big) = (c_1 f_2(\boldsymbol{x}) - c_1 f_2(\boldsymbol{r})),\ \mathsf{Enc}(c_1 f_2(\boldsymbol{r})))$, *ii)*
- f_1 const., f_2 quad.: $(c_1, \bot) \otimes \big(\bot,\ \mathsf{Enc}(f_2(\boldsymbol{x}) - f_2(\boldsymbol{r}))\big) = \big(\bot,\ \mathsf{Enc}(c_1 f_2(\boldsymbol{x}) - c_1 f_2(\boldsymbol{r}))\big)$, *iii)*
- f_1, f_2 linear: $\big(f_1(\boldsymbol{x}) - f_1(\boldsymbol{r}),\ \mathsf{Enc}_{\mathsf{pk}}(f_1(\boldsymbol{r}))\big) \otimes \big(f_2(\boldsymbol{x}) - f_2(\boldsymbol{r}),\ \mathsf{Enc}(f_2(\boldsymbol{r}))\big)$
  $= \big(\bot,\ (f_1(\boldsymbol{x}){-}f_1(\boldsymbol{r}))(f_2(\boldsymbol{x}){-}f_2(\boldsymbol{r})) \oplus ((f_1(\boldsymbol{x}){-}f_1(\boldsymbol{r})) \otimes \mathsf{Enc}(f_2(\boldsymbol{r}))) \boxplus ((f_2(\boldsymbol{x}){-}f_2(\boldsymbol{r})) \otimes \mathsf{Enc}(f_1(\boldsymbol{r})))\big)$
  $= \big(\bot,\ \mathsf{Enc}(f_1(\boldsymbol{x})f_2(\boldsymbol{x}) - f_1(\boldsymbol{r})f_2(\boldsymbol{r}))\big)$, *iii)*

Fig. 7. Computation of $eval_f(\boldsymbol{x} - \boldsymbol{r}, \mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{r}))$ for the induction step, with $f(\boldsymbol{x}) = f_1(\boldsymbol{x}) \diamond f_2(\boldsymbol{x})$.

and $m_1 := r$, answering with (m_0, m_1) as its first output in the IND-CPA game. Now $\mathcal{B}$'s challenger will sample a bit $b \xleftarrow{\$} \{0,1\}$ and respond with $\mathsf{ct} := \mathsf{Enc}(\mathsf{pk}, m_b)$. On receiving ct, $\mathcal{B}$ sends $(x - r, \mathsf{ct})$ to $\mathcal{A}$, who will eventually reply back with a bit b'. Finally, $\mathcal{B}$ outputs b'.

Observe that, if $b = 1$, ct contains the encryption of r. On the other hand, if $b = 0$, ct is the encryption of an independent random value s. Actually, note that for every $x \in \mathcal{M}$, the following two distributions are identical:

$$s, r \xleftarrow{\$} \mathcal{M};\ (x{-}r, \mathsf{Enc}(\mathsf{pk}, s)) \ \equiv\ y, r \xleftarrow{\$} \mathcal{M};\ (y, \mathsf{Enc}(\mathsf{pk}, r))\ .$$

Therefore, when $b = 1$, $\mathcal{B}$ is perfectly simulating the *masked experiment* in the Msk-vs-Rnd game, while when $b = 0$, $\mathcal{B}$ is perfectly simulating the *random experiment*. We conclude that $\mathcal{B}$ wins the IND-CPA game if and only if $\mathcal{A}$ wins the Msk-vs-Rnd game. □

Proof of Lemma 2.

We will prove the following stronger statement, which clearly implies Lemma 2:

Lemma 4. *Let f be quadratic circuit with variables x_i, $\forall i \in [\ell]$ and let $\boldsymbol{x}, \boldsymbol{r} \in \mathcal{M}^\ell$. Let (a, b) be the result of $eval_f(\boldsymbol{x}{-}\boldsymbol{r},\ \mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{r}))$. It holds:*

i) if $(a, b) \in \mathcal{M} \times \{\bot\}$, then $a = f(\boldsymbol{x})$.

ii) if $(a, b) \in \mathcal{M} \times \mathcal{C}$, $a = f(\boldsymbol{x}) - f(\boldsymbol{r}) \wedge \mathsf{Dec}_{\mathsf{sk}}(b) = f(\boldsymbol{r})$,

iii) if $(a, b) \in \{\bot\} \times \mathcal{C}$, then $\mathsf{Dec}_{\mathsf{sk}}(b) = f(\boldsymbol{x}) - f(\boldsymbol{r})$,

where the evaluation of f takes place over set Γ with operations $\oplus$ for addition and $\otimes$ for multiplication.

We prove the lemma by induction on the circuit structure. The base case is a circuit f with no gates (one single wire that is at the same time input and output). The $eval_f$ procedure does nothing, and the statement of the lemma holds immediately, since $f(x) = x$.

Now, take a general circuit f and express it as $f(x) = f_1(x) \diamond f_2(x)$, where $\diamond \in \{\oplus, \otimes\}$ is the last gate of circuit f. By the induction hypothesis, the lemma holds for circuits f_1 and f_2. We just need to show that it also holds for f. This calculation is done in Figure 7, where we leverage the homomorphic property of $\oplus$, $\otimes$ and $\boxplus$. Since we cover all relevant cases (f is quadratic, so there will be no multiplications between linear and quadratic terms or between two quadratic terms), this completes the proof. □

Proof of Lemma 3.

The algorithm proceeds as follows. First it uses $\mathsf{sk}_1^{\mathcal{E}}$ to decrypt $\mathsf{e}^{(1)}$, i.e., it computes $(a, b, \lambda) := \mathcal{E}.\mathsf{Dec}(\mathsf{sk}_1^{\mathcal{E}}, \mathsf{e}^{(1)})$. Second, it samples a random $r \xleftarrow{\$} \mathcal{M}$, and computes:

$$\begin{aligned} b' &:= r \oplus \mathcal{H}.\mathsf{AddPk}_{\mathsf{sk}_0^{\mathcal{H}}}(b) \\ \widetilde{\mathsf{e}}^{(0)} &:= \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}_0^{\mathcal{E}}}((a-r, b', \lambda)) \\ \widetilde{\mathsf{e}}^{(1)} &:= \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}_1^{\mathcal{E}}}((r, b, \lambda)) \end{aligned}$$

Let us now show that $(r_n, \widetilde{\mathsf{e}}^{(0)}, \widetilde{\mathsf{e}}^{(1)}, \mathsf{e}^{(2)}, \ldots, \mathsf{e}^{(n)})$ generated in this way is identically distributed to an honest encryption of x for $n+1$ authorities, in which the new authority is in the first position, and it is followed by the original authorities (in the same order).

This follows from two properties. First, by correctness of $\mathcal{H}.\mathsf{AddPk}$ and the circuit privacy of $\mathcal{H}$ we have that b' is identically distributed to a fresh encryption of $r + r_1$, where r_1 represents the randomness encrypted inside b. Second, by doing a simple change of variables, we can define $r \mapsto r_0 - r_1$, for a random $r_0 \xleftarrow{\$} \mathcal{M}$. This way, $(a - r, b')$ and (r, b) are identically distributed to $(r_0 - x, \mathsf{Enc}(r_0))$ and $(r_0 - r_1, \mathsf{Enc}(r_1))$, as required. □