HIKE: Walking the Privacy Trail *

Elena Pagnin, Carlo Brunetta, and Pablo Picazo-Sanchez

Chalmers University of Technology, Gothenburg, Sweden
{pagnin,brunetta,pablop}@chalmers.se

Abstract. We consider the problem of privacy-preserving processing of outsourced data in the context of user-customised services. Clients store their data on a server. In order to provide user-dependent services, service providers may ask the server to compute functions on the users' data. We propose a new solution to this problem that guarantees data privacy (*i.e.,* an honest-but-curious server cannot access plaintexts), as well as that service providers can correctly decrypt only –functions on– the data the user gave them access to (*i.e.,* service providers learn nothing more than the result of user-selected computations).
Our solution has as base point a new secure labelled homomorphic encryption scheme (LEEG). LEEG supports additional algorithms (FEET) that enhance the scheme's functionalities with extra privacy-oriented features. Equipped with LEEG and FEET, we define HIKE: a lightweight protocol for private and secure *storage*, *computation* and *disclosure* of users' data. Finally, we implement HIKE and benchmark its performances demonstrating its succinctness and efficiency.

Keywords: Homomorphic encryption, Privacy-preserving computation, Security protocol, GDPR

1 Introduction

We are living in the digital era, where people like to store their personal data in the cloud and get access to it any-time and anywhere. On the other hand, database maintainers and service providers develop an increasing interest for processing and extracting statistics from users's data. The usual setting is depicted in Figure 1: users (or clients) agree to share their personal data with some service providers which, in exchange, returns customised services and improved user-dependent performances. Typical application scenarios are: e-Health environments (*e.g.,* keeping a blood pressure database that doctors can access to retrieve data) or smart trackers (*e.g.,* activity bands that keep track of users' performance and achieved goals).
In recent years, the cryptographic community has proposed new techniques for computing on outsourced data including Fully Homomorphic Encryption [12], Verifiable Computation [10] or Multi-Key Homomorphic Signatures [8]. Beyond

* This work was partially supported by the the Swedish Research Council (*Vetenskapsrådet*) through the grants PolUser (2015-04154) and PRECIS (621-2014-4845).

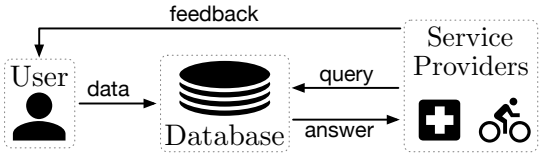


Fig. 1: The setting we consider: users send data to a database and enjoy some service. *Example 1* (e-Health): doctors can query for the average blood pressure in the last hour and alert the user in case of need. *Example 2* (sport): the service provider can query for the distance run until 'now' and feedback when the daily goal is achieved.

the obvious benefits, user-customised services may have undesirable drawbacks. In particular, service providers can collect data from thousands of clients, identify trends, profile users, and potentially sell their knowledge to third parties without the clients' consent or awareness.

In this paper, we define a model for user-customised services that addresses new privacy challenges inspired to the guidelines provided in the European *General Data Protection Regulation (GDPR)* [6]. This regulation sets clear boundaries on how data should be *collected, handled* and *processed* by protecting clients from possible miss-usages of their data by malicious service providers.

In particular, we give one of the *first attempts*[1] to *rigorously* formalise in *cryptographic* terms three of the main guidelines in the GDPR [6], namely: (*i*) the client's data is never stored in plaintext on public databases (art. 32); (*ii*) the client decides who can read her data (art. 15); (*iii*) the client has the *right to be forgotten*, *i.e.,* to request deletion of her data (art. 17).

Our Contributions. Our main contribution is the proposal and efficient instantiation of HIKE, a new cryptographic protocol that solves the problem of providing client-customised services. In details, our contributions are as follows:

(a) We present LEEG, a new labelled encryption scheme based on the elliptic-curve ElGamal scheme which supports homomorphic computation of multivariate linear polynomials.
(b) We define a set of additional algorithms that increase the versatility of LEEG, including an algorithm to cryptographically *destroy* encrypted data and a new procedure through which a chosen third party gets *decryption rights* for specific computations on encrypted data. We call this set of algorithms FEET as they extend the LEEG scheme.
(c) We then use LEEG and FEET in our HIKE protocol. HIKE is a novel lightweight protocol designed for application scenarios that involve users, servers, and service-providers. What makes this scenario different is that users' data need to be both privately and securely stored while allowing service providers to perform simple statistics on specific portions of users' data.
(d) We prove that HIKE is secure with respect to our security model that includes notions that address three articles of the GDPR law, namely (*i*) user's data is *never* stored as *plaintext* in the server; (*ii*) the user has the power to

[1] The only academic works we found related to the GDPR are [5,20], where the focus is on *technical* and *implementation* requirements. We could not find any work attempting to formalise and analyse the GDPR requirements in cryptographic terms.

decide *who* can *read* its data; (*iii*) the user can always ask the server to cryptographically destroy its data.

(e) We implement the HIKE protocol and empirically test its succinctness and efficiency. We provide a complete benchmark for all the algorithms involved. Our implementation is freely available at `https://github.com/Pica4x6/HIKE`.

Overview of our Technique. Our starting point is the ElGamal encryption scheme on elliptic curves [16,14]. We progressively change this scheme by introducing three *ideas*: (i) replacing the sampling of randomness in a ciphertext by using labels and *Pseudo Random Function (PRF)*; (ii) modifying the labels to include the public key of the scheme, and; (iii) exploiting the structure of the new ciphertexts to define algorithms for special user-privacy oriented features.

In more detail, but still quite abstractly, the three ideas work as follows. A label is a *unique identifier* for a specific message and it contains the sender's *public key*, a *random curve point* and a *tag* that identifies the message. Idea (i) is to change *how* the randomness is generated during the encryption procedure. We replace the random sampling of ElGamal encryption with the evaluation of a secure pseudo-random function $\mathsf{PRF}_{\mathsf{k}}$ on the label. For this change to work correctly, we also need to add the PRF key k to the user's secret key. The major implications of this change are: 1) we can get rid of the random component of classical ElGamal ciphertexts (thus achieving better succinctness), and 2) the new scheme has secret-key encryption.

Idea (ii) exploits the special structure of the labels and views the *"random curve point"* as the public key of the *designated-receiver* (*e.g.,* the service provider). By doing so, we can algebraically manipulate ciphertexts in meaningful ways and also allow data decryption for both the encryptor and the *designated-receiver* (the latter upon receiving a special data-dependent *token*).

The last idea (iii) is to combine (i) and (ii) and design a protocol which addresses: data-secrecy (similar notion to semantic security); token-secrecy (data owner have full control on *who* can decrypt their data), and; forgettability (data owners can ask for their data to be destroyed).

Related Work. Rivest *et al.* [23] introduced the concept of *Homomorphic Encryption (HE)* schemes as a set of algorithms that can be used to encrypt data, perform some computations on the ciphertexts, and directly decrypt the result of the computation.

For over 30 years, all secure proposals of HE schemes were only partially homomorphic, *i.e.,* they supported either additions or multiplications of ciphertexts [19,7]. The breakthrough result was due to Gentry [12] and started an avalanche of *Fully Homomorphic Encryption (FHE)* schemes [25,3,24,4]. However, most FHE schemes have major drawbacks due to key sizes and (or) efficiency. Albeit HE supports less expressive computations than FHE, as long as we are interested in simple statistics (*e.g.,* average, additions, least square fit of functions) on encrypted data, HE has better performances than FHE.

Barbosa *et al.* [2] introduced the notion of *Labelled Homomorphic Encryption (LabHE)* which combines HE with labels. This is an elegant approach to

address the problem of privacy-preserving processing of outsourced data. In this paper, we follow their definitional framework but we avoid the presence of a *fully trusted party* that executes the initial setup and holds a master secret key. Albeit being less expressive than Barbosa *et al.*'s scheme, our protocol achieves full succinctness without relaying on any trusted party.

A concurrent and independent work by Fischer *et al.* [9] proposes a linearly homomorphic construction also based on ElGamal encryption scheme. The aim of [9] is to provide both information flow security and authentication while our scheme has a privacy-oriented cryptographic approach —since we do not consider authentication, and it achieves full ciphertext succinctness.

2 Preliminaries

Notation. For any finite set S, we denote by $x \xleftarrow{\$} S$ the uniformly random sampling of elements from S, and by $|S|$ as the size of the set. We denote by $[n]$ the set $\{1, \dots, n\}$, by $[0..q]$ the set $\{0, \dots, q\}$, and by $\{0,1\}^*$ the space of binary-strings of arbitrary length. For any linear function f on n variables we describe f as $f(x_1, \dots, x_n) = a_0 + \sum_{i \in [n]} a_i x_i$, for opportune values $a_i \in [0..q-1]$. We denote by λ the security parameter of cryptographic schemes and functions, and by ε a negligible function in λ, *i.e.,* $\varepsilon(\lambda) = O(\lambda^{-c})$ for every constant $c > 0$. We refer to computational feasibility (resp. infeasibility) of a problem if all known algorithm to solve the problem run in polynomial (resp. exponential) time.

Elliptic Curves. For prime p, let $\mathcal{E}$ be an elliptic curve over $\mathbb{F}_p$ and P be a generator point for the group $\mathbb{G}$ derived by $\mathcal{E}$. Let q be the order of $\mathbb{G}$, *i.e.,* $\mathbb{G} =< P >= \{\mathcal{O}, P, 2 \cdot P, \dots, (q-1) \cdot P\}$, where $\mathcal{O}$ is the point at infinity (identity element of $\mathbb{G}$). For security reasons, we require q to be a prime number or a non-smooth (*i.e., q* is divisible by a large prime).

Problem 1 (Elliptic Curve Discrete Logarithm Problem [16]). Let p be a prime number and $\mathcal{E}$ be an elliptic curve over $\mathbb{F}_p$. Let $\mathbb{G}$ be the subgroup generated by a point $P \in \mathcal{E}$ such that $\mathbb{G} =< P >$ and $|\mathbb{G}| = q$ is prime or a non-smooth number. Given $Q \in \mathbb{G}$, the *Discrete Logarithm (DLog)* requires to find the value $m \in [0, \dots, q-1]$ such that $m \cdot P = Q$.

Pollard's Rho [21] is a well-known algorithm for solving the DLog problem. Its running time, however, is exponential in the group size, *i.e.,*$O(\sqrt{|\mathbb{G}|}) = O(2^{\frac{q}{2}})$.

Assumption 1 *Given $\mathbb{G}$, P and Q as in Problem 1, it is computationally infeasible to find a solution to the DLog.*

Problem 2 (Interval Discrete Logarithm Problem [22]). Let $\mathcal{E}$, $\mathbb{F}_p$, $\mathbb{G}$, P and Q be as in Problem 1. The *Interval Discrete Logarithm Problem (IDLP)* requires to find the value $m \in [0, \dots, q-1]$ such that $m \cdot P = Q$ knowing that $m \in [a, \dots, b]$ for $a, b \in [0, \dots, q-1]$.

Pollard's kangaroo algorithm [22] finds an existing solution to the IDLP problem in a given interval $[a, \dots, b]$ in time $O(2^{\frac{\Delta}{2}})$ where $\Delta = \lceil \log_2(b-a) \rceil$ is the number of bits in the binary representation of the interval length [17].

Assumption 2 *Solving the IDLP is computationally feasible for* $|[a..b]| < 2^{22}$*, while it is infeasible for larger intervals* $|[a..b]| > 2^{160}$*.*

Pseudo Random Functions (PRF) [15]. A PRF is a collection of keyed functions from a (possibly infinite) set A to a finite set B. Formally, let $\mathcal{F}$ be the set of all functions from A to B and $\mathcal{K}$ be a (finite) set of keys, a PRF family is a set of functions $\{\mathsf{PRF}_\mathsf{k} : A \to B \mid \mathsf{k} \in \mathcal{K}\}$ satisfying the following properties:

1. For any $a \in A$ and $\mathsf{k} \in \mathcal{K}$, the function $\mathsf{PRF}_\mathsf{k}(a)$ is efficiently computable.
2. No *Probabilistic Polynomial-Time (PPT)* algorithm can distinguish the function PRF_k (for $\mathsf{k} \xleftarrow{\$} \mathcal{K}$) from a function $f \xleftarrow{\$} \mathcal{F}$.

In this paper, we regard HMAC-SHA256 as secure pseudo random function family for functions $\mathsf{PRF}_\mathsf{k} : \{0,1\}^* \to \mathbb{Z}_q$ with $\mathsf{k} \in \mathcal{K}$.

2.1 Labelled Homomorphic Encryption

The notion of labelled homomorphic encryption was introduced by Barbosa *et al.*to improve the efficiency of HE schemes [2]. The main idea is to combine homomorphic encryption [12] with labelled programs [11] to be able to compute on selected outsourced ciphertexts. A labelled program $\mathcal{P}$ is a tuple $(f, (\ell_1, \ldots, \ell_n))$, such that $f : X^n \to X$ is a function of n variables and ℓ_i is a label for the i-th input of f. Labelled programs can be used to identify users' input to computations by imposing $\ell = (\mathsf{id}, \tau)$ for some user identifier id and tag τ [8]. We denote by $\mathcal{I}_\ell = (f, \ell)$ the *identity labelled program* on the label ℓ, *i.e.,* $f_\ell(x) = x$.

Formally, a labelled homomorphic encryption scheme $\mathsf{LabHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ is defined by the following algorithms:

$\mathsf{KeyGen}(1^\lambda)$**:** on input the security parameter, it outputs a secret key sk and a (public) evaluation key ek that includes a description of a message space $\mathcal{M}$, a label space $\mathcal{L}$, and a class of admissible functions $\mathcal{F}$.

$\mathsf{Enc}(\mathsf{sk}, \ell, m)$**:** on input sk, a label ℓ, and a message m, it outputs a ciphertext ct.

$\mathsf{Eval}(\mathsf{ek}, f, \mathsf{ct}_1, \ldots, \mathsf{ct}_n)$**:** on input ek, a function $f : \mathcal{M}^n \to \mathcal{M}$ in a set of admissible functions $\mathcal{F}$, and n ciphertexts. It returns a ciphertext ct.

$\mathsf{Dec}(\mathsf{sk}, \mathcal{P}, \mathsf{ct})$**:** on input sk, a labelled program $\mathcal{P} = (f, \ell_1, \ldots, \ell_n)$ and a ciphertext ct, it outputs a message m.

Moreover, LabHE satisfies the properties of correctness, succinctness, (semantic) security and context hiding defined in as follows.

Definition 1 (Correctness [2]). *A* LabHE *scheme is said to be correct for a family of functions* $\mathcal{F}$ *if, for all keys* $(\mathsf{ek}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$*, all* $f \in \mathcal{F}$*, any selection of labels* $\ell_1, \ldots, \ell_n \in \mathcal{L}$*, and messages* $m_1, \ldots, m_n \in \mathcal{M}$*, with corresponding ciphertexts* $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{sk}, \ell_i, m_i)$*,* $i \in [n]$*, and* $\mathcal{P} = (f, (\ell_1, \ldots, \ell_n))$*, it holds that:*

$$\Pr\left[\mathsf{Dec}\left(\mathsf{sk}, \mathcal{P}, \mathsf{Eval}(\mathsf{ek}, f, \mathsf{ct}_1, \ldots, \mathsf{ct}_n)\right) = f(m_1, \ldots, m_n)\right] \geq 1 - \varepsilon.$$

Definition 2 (Succinctness [2]). *A* LabHE *scheme is said to be succinct if there exists a fixed polynomial* $\mathsf{poly}(\cdot)$ *such that every honestly generated ciphertext (output by* Enc *or* Eval*) has bit-size size* $\mathsf{poly}(\lambda)$*.*

The security notion for LabHE schemes is inspired to the standard semantic security experiment proposed by Goldwasser and Micali [13].

Definition 3 (Context Hiding [2]). *A* LabHE *scheme is context-hiding if there exists a PPT algorithm* $\mathcal{S}$ *such that, for any* $(\mathsf{ek}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, $f \in \mathcal{F}$, *any tuple of labels* $\ell_1, \ldots, \ell_n \in \mathcal{L}$ *and messages* $m_1, \ldots m_n \in \mathcal{M}$ *with corresponding ciphertexts* $\mathsf{ct}_1 \leftarrow \mathsf{Enc}(\mathsf{sk}, \ell_i, m_i)$, *if* $m = f(m_1, \ldots, m_n)$ *and* $\mathcal{P} = (f, \ell_1, \ldots, \ell_n)$ *then:*

$$\tfrac{1}{2} \cdot \sum_{\mathsf{ct}} \left| \mathsf{Prob}[\mathsf{Eval}(\mathsf{ek}, f, \mathsf{ct}_1, \ldots, \mathsf{ct}_n) = \mathsf{ct}] - \mathsf{Prob}[\mathcal{S}(1^\lambda, \mathsf{sk}, \mathcal{P}, m) = \mathsf{ct}] \right| < \varepsilon(\lambda).$$

Definition 4 (Semantic security for LabHE [2]). *A* LabHE *scheme is semantically secure if for any PPT algorithm* $\mathcal{A}$ *taking part to* $\mathtt{Exp}^{\mathsf{sem.sec}}_{\mathsf{LabHE},\mathcal{A}}$ *in Figure 2, it holds that:* $\mathtt{Adv}^{\mathsf{sem.sec}}_{\mathsf{LabHE},\mathcal{A}}(\lambda) = \Pr\left[\mathtt{Exp}^{\mathsf{sem.sec}}_{\mathsf{LabHE},\mathcal{A}}(\lambda) = 1\right] - \frac{1}{2} < \varepsilon$.

$\underline{\mathtt{Exp}^{\mathsf{sem.sec}}_{\mathsf{LabHE},\mathcal{A}}(\lambda)}$:

$b \xleftarrow{\$} \{0,1\}, L_{\mathsf{lab}} = \emptyset$
$(\mathsf{ek}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
$(\ell^*, m_0, m_1) \leftarrow \mathcal{A}^{O\mathsf{Enc}_{\mathsf{sk}}}(\mathsf{ek})$
if $\ell^* \in L_{\mathsf{lab}}$
  $\mathsf{ct} = \mathtt{error}$
else
  $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{sk}, \ell^*, m_b)$
  $L_{\mathsf{lab}} \leftarrow L_{\mathsf{lab}} \cup \ell^*$
$b' \leftarrow \mathcal{A}^{O\mathsf{Enc}_{\mathsf{sk}}}(\mathsf{ct})$
Output $\mathtt{1}$ if $b = b'$, and $\mathtt{0}$ otherwise.

$O\mathsf{Enc}_{\mathsf{sk}}(\ell, m)$:
if $\ell \in L_{\mathsf{lab}}$
  return $\mathtt{error}$
else $L_{\mathsf{lab}} \leftarrow L_{\mathsf{lab}} \cup \ell$
  return $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{sk}, \ell, m)$.

Fig. 2: The semantic-security experiment for LabHE schemes, and the OEnc oracle.

3 Labelled Elliptic-curve ElGamal (LEEG).

In its standard construction, the ElGamal encryption scheme [7] is defined on finite multiplicative groups and is only multiplicative-homomorphic. It is possible to obtain an additive-homomorphic version of ElGamal by using groups defined over an elliptic curve [16,14] and specific message encoding maps, discussed later in Section 7. Essentially, in the elliptic curve setting, exponentiations are replaced by multiplications and multiplications by additions. Security reduces to the hardness of computing the discrete logarithm on elliptic curves. For further details on ElGamal for elliptic curve groups see [14].

In this section, we define the first labelled and symmetric-key version of the additive-homomorphic ElGamal scheme that we refer to as LEEG (Labelled Elliptic-curve ElGamal). To ease the adoption of LEEG in our GDPR-oriented protocol HIKE in Section 5.1, we make a small adaptation to Barbosa *et al.*'s framework for LabHE. We introduce a SetUp algorithm that outputs some global public parameters pp, and make the KeyGen algorithm run on pp. This change

is only syntactic if KeyGen is run once and brings with straightforward modifications to the definitions.

Definition 5 (LEEG). *The* LEEG *scheme is defined by the following five PPT algorithms:*

$\mathsf{SetUp}(1^\lambda)$**:** *on input the security parameter , the setup algorithm outputs* pp *that include: a λ-bit-size prime p, an elliptic curve $\mathcal{E}$ over $\mathbb{F}_p$ with a (prime) order-q group $\mathbb{G} \subseteq \mathcal{E}$, a generator P of the group $\mathbb{G}$, a set of admissible functions $\mathcal{F}$ (namely linear functions), a set of messages $\mathcal{M} \in [m]$, a set of message identifiers $\mathcal{T} = \{0,1\}^t$, a set of labels $\mathcal{L} = \mathbb{G} \times \mathbb{G} \times \mathcal{T}$, and a keyed-pseudo-random function family* PRF *from $\{0,1\}^*$ to $[0..q-1]$. The* pp *are input to all subsequent algorithms even if not stated explicitly.*

$\mathsf{KeyGen}(\mathsf{pp})$**:** *on input the public parameters the key generation algorithm selects a random element $sk \xleftarrow{\$} [q-1]$ and a random PRF key $\mathsf{k} \xleftarrow{\$} \mathcal{K}$. It outputs the secret key $\mathsf{sk} = (sk, \mathsf{k})$.*[2]

$\mathsf{Enc}(\mathsf{sk}, \ell, m)$**:** *on input a secret key $\mathsf{sk} = (sk, \mathsf{k})$, a label $\ell = (sk \cdot P, Q, \tau)$ with $Q \in \mathbb{G}$ and message $m \in \mathcal{M}$ the encryption algorithm returns the ciphertext:*

$$\mathsf{ct} = m \cdot P + \mathsf{PRF}_{\mathsf{k}}(\ell) \cdot sk \cdot Q \in \mathbb{G} \tag{1}$$

In case the input label has as first entry a value different from $sk \cdot P$ the algorithm returns `error`*.*

$\mathsf{Eval}(f, \mathsf{ct}_1, \ldots, \mathsf{ct}_n)$**:** *on input a linear function $f(x_1, \ldots, x_n) = a_0 + \sum_{i \in [n]} a_i x_i$ and n ciphertexts ct_i, the evaluation algorithm returns the ciphertext:*

$$\mathsf{ct} = a_0 \cdot P + \sum_{i \in [n]} a_i \cdot \mathsf{ct}_i \in \mathbb{G} \tag{2}$$

$\mathsf{Dec}(\mathsf{sk}, \mathcal{P}, \mathsf{ct})$**:** *on input a secret key $\mathsf{sk} = (sk, \mathsf{k})$, a labelled program $\mathcal{P} = (f, \ell_1, \ldots, \ell_n)$ for a linear function f with labels of the form $\ell_i = (sk \cdot P, Q_i, \tau_i)$, and a ciphertext* ct*, the decryption algorithm computes:*

$$T = \mathsf{ct} - sk \cdot \Big(\sum_{i \in [n]} a_i \cdot \mathsf{PRF}_{\mathsf{k}}(\ell_i) \cdot Q_i \Big) \in \mathbb{G} \tag{3}$$

and returns $m = \log_P(T)$.

We note that LEEG is a fully dynamic scheme, indeed ciphertexts output by Eval can be used as input to new computations (as long as the new computation includes the initial labelled program).

Succinctness of LEEG. The succinctness of the LEEG scheme is immediate given that ciphertexts (output by Enc or Eval) are always one single group element in $\mathbb{G} \subseteq \mathcal{E}$. Further details regarding the actual bit-size for our implementation can be found in Section 7.

[2] In the original definition of Labelled Homomorphic Encryption [2], the KeyGen algorithm additionally outputs a public evaluation key. Since in our case this key is empty, we decided to skip it and have more succinct algorithm descriptions.

Correctness of LEEG. The correctness of LEEG is a straightforward computation.

Context-hiding of LEEG. The context-hiding property of LEEG is straightforward since given $\mathsf{sk}, \mathcal{P}$ and $m = f(\mathsf{ct}_1, \dots, \mathsf{ct}_n)$ the simulator is able to reconstruct exactly the same ciphertext output by $\mathsf{Eval}(f, \mathsf{ct}_1, \dots, \mathsf{ct}_n)$ as $\mathcal{S}(\mathsf{sk}, \mathcal{P}, m) := f(m_1, \dots, m_n) \cdot P + sk \sum_{i \in [n]} a_i \mathsf{PRF}_{\mathsf{k}}(\ell_i) \cdot Q_i$.

Security of LEEG. The proof is rather simple and we provide here just an intuition, deferring the formal proof to Appendix A. First the challenger replaces PRF with a truly random function, that is, the values r are now taken uniformly at random from $[q-1]$. At this point the security of the scheme becomes information theoretic.[3] We then prove the semantic security in the same way as for standard OTP-based schemes, *i.e.,* we show that the challenge ciphertext has the same probability of being an encryption of m_0 as of m_1.

4 FEET: Feature Extensions to the labelled-homomorphic El-gamal encrypTion scheme

In this section we define FEET a set of additional algorithms that increase the versatility and use cases of LEEG. The new algorithms build on the following observation. Given a label $\ell = (Q_1, Q_2, \cdot)$ we can interpret its first component Q_1 as the public key of the user who is performing the encryption (that we call this data-owner), and Q_2 as the public key of another user (that we call intended receiver). By doing so, we can give a sensible meaning to the procedures and manipulate ciphertexts in such a way that decryption works correctly only for data encryptor statde in the first component of the label, and the *designated-receiver* identified by the second component of the label. Assuming the existence of a *Public Key Infrastructure (PKI)*, FEET exploits the algebraic structure of LEEG ciphertexts to perform two actions:

- Cryptographically *'delete'* data owner's ciphertexts from a database by making them un-decryptable, *i.e.,* even the original data-owner would retrieve a random message by decrypting a *destroyed* ciphertext.
- Allow the data-owner to generate a special piece of information (called *token*) that enables the intended receiver to decrypt the output of a specific labelled program run on the encryptor's data.

Definition 6 (FEET: set of additional algorithms for LEEG). *Let* $\mathsf{LEEG} = (\mathsf{SetUp}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *be the labelled homomorphic encryption scheme of Definition 5, where the* KeyGen *algorithm is run multiple times and associates identities (identifiers* id*) to the keys it generates. We define:*

[3] Indeed, even provided an oracle access to an efficient solver of the discrete logarithm problem, the only information the adversary would retrieve from the challenge cipher-text is a random message $m' = m_b + r \cdot sk$. This is possible because, differently from El Gamal ciphertexts, LEEG ciphertexts have a single component that combines message and randomness.

$\mathsf{Destroy}(\mathsf{ct})$**:** *on input a ciphertext* ct*, the destroy-cihpertext algorithm picks a random* $r \xleftarrow{\$} [0..q-1]$ *and outputs the destroyed ciphertext* $\mathsf{ct}' = \mathsf{ct} + r \cdot P$*.*

$\mathsf{PublicKey}(\mathsf{sk})$**:** *on input the secret key* $\mathsf{sk} = (sk, \mathsf{k})$ *output the corresponding public key* $\mathsf{pk} = sk \cdot P$*.*

$\mathsf{TokenGen}(\mathsf{sk}, \mathcal{P})$**:** *on input a secret key* $\mathsf{sk} = (sk, \mathsf{k})$ *and a labelled program* $\mathcal{P} = (f, \ell_1, \ldots, \ell_n)$ *the token generation algorithm checks if the labels are of the form* $\ell_i = (sk \cdot P, Q, \tau_i)$ *— for a point* $Q \in \mathbb{G}$ *and some* $\tau_i \in \mathcal{T}$*,* $i \in [n]$*. If the condition is not satisfied, the algorithm returns* `error`*; otherwise, it parses* $f(x_1, \ldots, x_n) = a_0 + \sum_{i \in [n]} a_i x_i$ *and outputs :*

$$\mathsf{tok} = sk \cdot \Big(\sum_{i \in [n]} a_i \cdot \mathsf{PRF}_{\mathsf{k}}(\ell_i) \Big) \cdot P \tag{4}$$

$\mathsf{TokenDec}(\mathsf{sk}, \mathsf{ct}, \mathsf{tok})$**:** *on input a secret key* $\mathsf{sk} = (sk, \mathsf{k})$*, a ciphertext* ct *and a token* tok*, the decryption-with-token algorithm outputs* $m' = \log_P (\mathsf{ct} - sk \cdot \mathsf{tok})$*.*

Information theoretic security of $\mathsf{Destroy}$**.** We prove this property by showing that for any given message m and ciphertext ct', ct' is a possible 'destruction' of $\mathsf{ct} = \mathsf{Enc}(\mathsf{sk}, \ell, m)$ for any label. More formally, for any $m \in \mathcal{M}$, $\ell = (sk \cdot P, \mathsf{pk}, \tau)$ and $\mathsf{ct}' \in \mathbb{G}$ it holds that:

$$\mathsf{Prob}[\mathsf{Destroy}(\mathsf{Enc}(\mathsf{sk}, \ell, m)) = \mathsf{ct}'] = \frac{|\{r : \mathsf{ct}' = (m + r') \cdot P + sk \cdot r \cdot \mathsf{pk}\}|}{|\mathbb{G}|} = \frac{1}{|\mathbb{G}|}$$

where the probability is taken over all possible random choices in the $\mathsf{Destroy}$ algorithm ($r' \in [0..q-1]$), and $r = \mathsf{PRF}_{\mathsf{k}}(\ell)$. In particular, for any pair of label-message couples $(\ell_0, m_0), (\ell_1, m_1)$ it holds that:

$$\mathsf{Prob}[\mathsf{Destroy}(\mathsf{Enc}(\mathsf{sk}_{\mathsf{id}_0}, \ell_0, m_0)) = \mathsf{ct}'] = \mathsf{Prob}[\mathsf{Destroy}(\mathsf{Enc}(\mathsf{sk}_{\mathsf{id}_1}, \ell_1, m_1)) = \mathsf{ct}'].$$

Therefore given a ct' output by $\mathsf{Destroy}$ this could be generated by the ciphertext of *any* message $m \in \mathcal{M}$.

Correctness of $\mathsf{TokenDec}$**.** In order to prove the correctness of the decryption-with-token algorithm we need to show that

$$\mathsf{TokenDec}(\mathsf{sk}_2, \mathsf{Eval}(f, \mathsf{ct}_1, \ldots, \mathsf{ct}_n), \mathsf{TokenGen}(\mathsf{sk}_1, \mathcal{P})) = f(m_1, \ldots, m_n)$$

where $\mathsf{ct}_i = \mathsf{Enc}(\mathsf{sk}_1, \ell_i, m_i)$, $\ell_i = (\mathsf{pk}_1, \mathsf{pk}_2, \tau_i)$ for some $\tau_i \in \mathcal{T}$, and $\mathcal{P} = (f, \ell_1, \ldots, \ell_n)$. This is a straightforward computation and follows from Equation (2), Equation (4) and the fact that $\mathsf{pk}_i = sk_i \cdot P$.

Remark 1 (Composability of $\mathsf{TokenGen}$*).* It is possible to combine two (or more) decryption tokens $\mathsf{tok}_1, \mathsf{tok}_2$ generated for distinct labelled programs $\mathcal{P}_1, \mathcal{P}_2$, to obtain a joint decryption token tok that enables the intended decryptor with public key $\mathsf{pk}_2 = Q$ (common to all the labels involved) to correctly decrypt in one-shot the ciphertext for any labelled program $\mathcal{P}$ such that $f = b_1 f_1 + b_2 f_2$, for any $b_1, b_2 \in [0..q-1]$. Intuitively, this property follows by the linearity of the sum. A detailed explanation can be found in Appendix B.

5 The HIKE protocol

In this section, we introduce HIKE: a protocol that employs our LEEG scheme and its extra features FEET to achieve advanced properties relevant to real world applications.

In what follows, we present a use case for HIKE and defer the formal description to the Section 5.1. We consider a scenario with three types of actors: data-owners (called clients and denoted as C), a cloud server (denoted as S) that controls the database Δ where the clients' records are stored, and service providers (denoted as P). The work-flow of the interactions between these actors is depicted in Figure 3. As a use case, consider clients with smart-watches used for tracking their sport performances. With HIKE clients can safely upload their data on the cloud, cancel previously uploaded records, retrieve their data (or functions of thereof) at any time. Moreover, clients can allow their personal trainer app to access specific aggregations of data to provide personalised performance feedback.

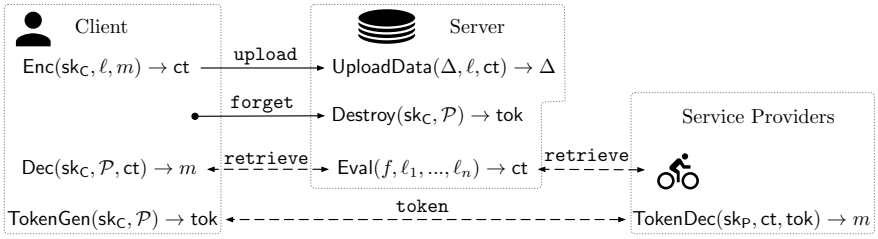


Fig. 3: Clients upload their encrypted data to the server (via an `upload` request). At any point in time, clients have the right to *destroy* their records in the database (via a `forget` request). In order to obtain aggregate information on the stored data, clients and service providers can ask the server to compute certain functions on the outsourced data and return the (encrypted) result (via a `retrieve` request). Finally, clients are always able to decrypt their own retrieved data, while service providers cannot decrypt directly. In order to decrypt the result of a computation $\mathcal{P} = (f, \ell_1, \ldots, \ell_n)$ on the client's data, the service provider needs to ask the client to generate a computation-specific decryption token (via a `token` request) that enables the designated service provider to decrypt.

5.1 A formal description

Definition 7 (The HIKE protocol). *Let* LEEG = (SetUp, KeyGen, Enc, Dec, Eval) *be the labelled homomorphic encryption scheme in Definition 5 enhanced with the algorithms* FEET = (TokenGen, TokenDec, Destroy) *described in Definition 6. We assume a PKI that, at every run of the* KeyGen *algorithm, associates identities (identifiers* id*) to the freshly generated keys. The* HIKE *protocols is defined by the following procedures:*

$\mathsf{Initialise}(1^\lambda)$: *on input the security parameter the initialisation procedure runs* $\mathsf{SetUp}(1^\lambda) \to \mathsf{pp}$ *and returns the public parameters. Implicitly, it also generates a database* Δ *and a public key infrastructure.*

$\mathsf{SignUp}(\mathsf{id})$: *on input a user identifier* id *the sign-up procedure returns* $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ *and updates the public ledger (PKI) with* $(\mathsf{id}, \mathsf{pk}_{\mathsf{id}})$ *where* $\mathsf{pk}_{\mathsf{id}} = \mathsf{PublicKey}(\mathsf{sk}_{\mathsf{id}})$. *For correctness, this procedure outputs* $\perp$ *if user* id *was already present in the system.*

$\mathsf{Encrypt}(\mathsf{sk}, \ell, m)$: *on input a secret key* sk*, a label* ℓ *and a message* m *the encryption procedure returns the ciphertext* $\mathsf{ct} = \mathsf{Enc}(\mathsf{sk}, \ell, m)$.

$\mathsf{UploadData}(\Delta, \ell, \mathsf{ct})$: *on input a database* Δ*, a label* ℓ *and a ciphertext* ct *the upload data procedure performs* $\Delta = \Delta \cup \{(\ell, \mathsf{ct})\}$.

$\mathsf{Forget}(\Delta, \ell)$: *on input a database* Δ *and a label* ℓ *the forget-ciphertext procedure retrieves the record* (ℓ, ct) *from* Δ *and replaces it with* (ℓ, ct') *where* $\mathsf{ct}' \leftarrow \mathsf{Destroy}(\mathsf{ct})$*, i.e.,it outputs* $\Delta = \Delta \setminus \{(\ell, \mathsf{ct})\} \cup \{(\ell, \mathsf{ct}')\}$.

$\mathsf{Compute}(\Delta, \mathcal{P})$: *on input a database* Δ *and a labelled program* $\mathcal{P} = (f, \ell_1, \dots, \ell_n)$ *the retrieve-data (or aggregate data) procedure collects the ciphertexts* ct_i *corresponding to labels* ℓ_i *present in* Δ *and returns* $\mathsf{ct} = \mathsf{Eval}(\mathsf{pp}, f, \mathsf{ct}_1, \dots, \mathsf{ct}_n)$.

$\mathsf{Decrypt}(\mathsf{sk}, \mathcal{P}, \mathsf{ct})$: *on input a secret key* sk*, a labelled program* $\mathcal{P}$*, and a ciphertext* ct *the decryption procedure outputs* $m = \mathsf{Dec}(\mathsf{sk}, \mathcal{P}, \mathsf{ct})$.

$\mathsf{AllowAccess}(\mathsf{sk}, \mathcal{P})$: *on input a user's secret key* sk *and a labeled program, the allow-access procedure returns* $\mathsf{tok} = \mathsf{TokenGen}(\mathsf{sk}, \mathcal{P})$.

$\mathsf{AccessDec}(\mathsf{sk}, \mathsf{ct}, \mathsf{tok})$ *on input a user's secret key* sk*, a ciphertext* ct *and a decryption token* tok*, the allowed-decryption procedure returns the output of* $\mathsf{TokenDec}(\mathsf{sk}, \mathsf{ct}, \mathsf{tok}) = m$.

5.2 Evaluation correctness of HIKE

The evaluation correctness of our HIKE protocol essentially reduces to the correctness of the underlying LEEG scheme (Definition 5) and FEET (Definition 6). Formally, the HIKE is correct if for any $\mathsf{pp} \leftarrow \mathsf{Initialise}(1^\lambda)$, for any combination of keys $(\mathsf{pk}_\mathsf{C}, \mathsf{sk}_\mathsf{C})$, $(\mathsf{pk}_\mathsf{P}, \mathsf{sk}_\mathsf{P})$ generated by the SignUp procedure, for any labelled program $\mathcal{P} = (f, \ell_1, \dots, \ell_n)$ with labels for the form $\ell = (sk_\mathsf{C} \cdot P, Q = \mathsf{pk}_\mathsf{P}, \cdot)$, for any set of messages $m_i \in \mathcal{M}$ with ciphertexts $\mathsf{ct}_i = \mathsf{Encrypt}(\mathsf{sk}, \ell_i, m_i)$, and for $m = f(m_1, \dots, m_n)$ it holds that:

(1) $\mathsf{Decrypt}(\mathsf{sk}_\mathsf{C}, \mathcal{P}, \mathsf{Compute}(\mathcal{P})) = m$.
(2) $\mathsf{AccessDec}\left(\mathsf{sk}_\mathsf{P}, \mathsf{Compute}(\mathcal{P}), \mathsf{AllowAccess}(\mathsf{sk}_\mathsf{C}, \mathcal{P})\right) = m$.

Condition *(1)* is equivalent to the evaluation correctness of the LEEG scheme given that the Compute procedure returns the output of LEEG's Eval algorithm and the Decrypt procedure runs LEEG's Dec algorithm.

Condition *(2)* is equivalent to the correctness of LEEG's additional algorithms given that AllowAccess returns the output of TokenGen, Compute returns the output of LEEG's Eval algorithm and AccessDec runs the TokenDec algorithm.

5.3 Interactions between the procedures of HIKE

We consider three categories of users taking part to the HIKE protocol:

Clients C: (or data owners), these users can run the procedures: SignUp, Encrypt, Decrypt and AllowAccess.

Server S: (or database maintainer), this user can run the procedures: UploadData, Compute and Forget.

Service providers P: (or third-party applications), these users can run the procedures: SignUp and AccessDec.

To simulate a real-world scenario, we allow users registered in the system to interact with each other. We model interaction via `requests` sent from one user to another and that there exists an authentication system to ensure this. Moreover, we assume that the target user reacts to the received request as follows:

`upload`**:** upload data requests can be performed by clients only and are directed to the server. Upon receiving an $\texttt{upload}(\ell, \mathsf{ct})$ request by a client C, the server checks if the submitted record is a new one , *i.e.,* if $(\ell, \cdot) \notin \Delta$. In this is the case, S runs $\mathsf{UploadData}(\Delta, \ell, \mathsf{ct})$ and returns `done` to C, otherwise S returns `error`.

`forget`**:** forget-ciphertext requests can be performed by clients only and are directed to the server. Upon receiving an $\texttt{forget}(\ell)$ request by a client C, the server checks that the label is a legit one for C, *i.e.,* that $\ell = (\mathsf{pk}_{\mathsf{C}}, \cdot, \cdot)$ and that $(\ell, \cdot) \in \Delta$. If both conditions holds, S runs $\mathsf{Forget}(\Delta, \ell)$ and returns `done` to C, otherwise it returns `error`.

`retrieve`**:** retrieve-data requests can be performed by clients or by service providers and are directed to the server. Upon receiving a $\texttt{retrieve}(\mathcal{P})$ request the server checks if the labelled program $\mathcal{P} = (f, \ell_1, \dots, \ell_n)$ is well-defined, *i.e.,* if for every $i \in [n]$, $(\ell_i, \cdot) \in \Delta$ and $\ell_i = (\mathsf{pk}_h, \mathsf{pk}_k, \cdot)$ for some users registered in the systems. If the conditions hold, S runs $\mathsf{Compute}(\Delta, \mathcal{P}) = \mathsf{ct}$ and returns ct to whom performed the query, otherwise it returns `error`.

`token`**:** access-token requests can be performed by service providers only and are directed to clients only. Upon receiving a $\texttt{token}(\mathcal{P})$ request, the client has the freedom to decide whether to reply consistently, running $\mathsf{AllowAccess}(\mathsf{sk}, \mathcal{P}) = \mathsf{tok}$ and returning tok to P, or to ignore the query returning `error`.

6 Security model and proofs for HIKE

Our security model builds on the setting introduced in Section 5.3 and covers three main goals:

1) data-secrecy, *i.e.,* confidentiality of the clients' data;
2) token-secrecy, *i.e.,* clients have full control on *who* can decrypt their data (only targeted service providers can decrypt, and no one else); and
3) forgettability, *i.e.,* clients can ask for their data to be destroyed.

Interestingly, these security notions cover three of the requirements presented in the GDPR: the confidentiality of personal data (security of processing, art. 32), the clients' right of access (and share) data (art. 15), and; the right to ask for erasure of her personal data (*right to be forgotten*, art. 17) [6].

Adversarial model We denote malicious users with the user's category and the symbol *, *e.g.,* P^*, and make the following assumptions:

- Clients C_i are *honest*, *i.e.,* they behave according to the interactions described in Section 5.3.
- The server S is *honest but curious*, *i.e.,* it behaves according to the interactions in Section 5.3 but tries to infer information about the clients' data (passive adversary).[4]
- Service Providers P_j can be *malicious* and deviate from the protocol in arbitrary ways.

We note that since anyone can generate and register keys in the protocol (using the PKI infrastructure), a malicious server corresponds to a malicious service provider that has access to an honest server (as the latter would reply to any `retrieve` request).

6.1 Data secrecy

Our notion of data-secrecy is inspired to the definition of semantic-security for labelled homomorphic encryption by Barbosa *et al.* [2] but adapted to our protocol's setting. Intuitively, we require that the adversary $\mathcal{A}$, who controls a (malicious) server provider P^* and holds a copy of the (encrypted) database Δ, should not be able to determine the plaintext associated to a database record (ℓ, ct). We formalise the notion through the data-secrecy experiment in Figure 4.
Notably, $\mathcal{A}$ is given access to three oracles: $O\mathsf{SignUp}$ to simulate users registering to the system, $O\mathsf{Encrypt}$ to populate the database with adversarial chosen data (*i.e.,* $\mathcal{A}$ chooses the messages encrypted by a client). With abuse of notation we will write $\mathsf{pk}_{\mathsf{id}} \in L_{\mathsf{keys}}$ meaning that L_{keys} has an element of the form $(\mathsf{id}, \mathsf{sk}_{\mathsf{id}}, \mathsf{pk}_{\mathsf{id}})$.

Theorem 1. *The* HIKE *protocol achieves data-secrecy, i.e., for any PPT adversary $\mathcal{A}$ taking part to the experiment in Figure 4, it holds that:*

$$\mathtt{Adv}^{\mathsf{data.sec}}_{\mathsf{HIKE},\mathcal{A}}(\lambda) = \Pr\left[\mathtt{Exp}^{\mathsf{data.sec}}_{\mathsf{HIKE},\mathcal{A}}(\lambda) = 1\right] - \frac{1}{2} \leq Q_{\mathsf{id}} \cdot \mathtt{Adv}^{\mathsf{sem.sec}}_{\mathsf{LEEG},\mathcal{A}}(\lambda),$$

where Q_{id} is a bound on the total number calls to the sign-up oracle.

Proof. We exhibit a reduction $\mathcal{B}$ that uses $\mathcal{A}$ to win the semantic-security experiment for the LEEG scheme. At the beginning the reduction samples $\mathsf{id}^\star \in \mathsf{ID}$ as its guess for the identity that $\mathcal{A}$ will target during the game. (Note that

[4] This assumption removes the theoretical need for the definition of forgettability in our security model. We include it for completeness.

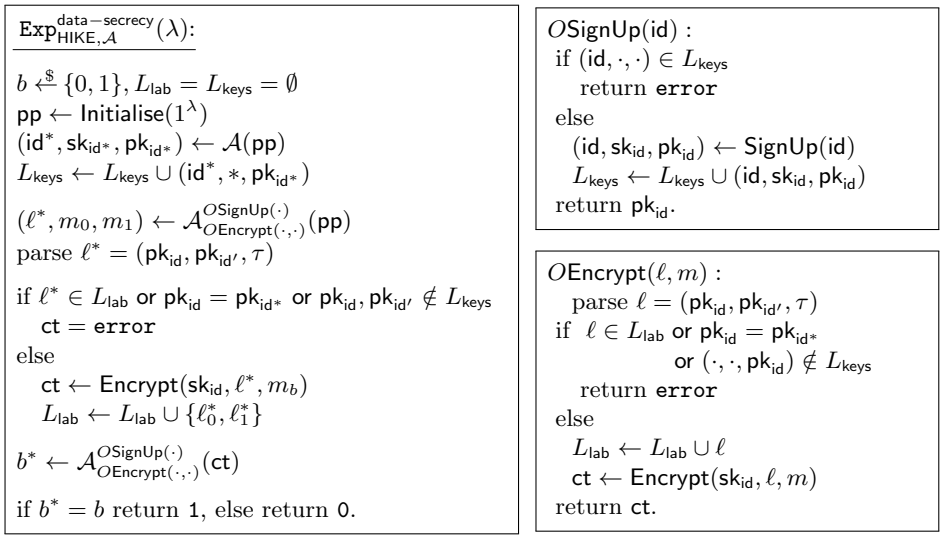


Fig. 4: The data-secrecy experiment and the oracles $O\mathsf{SignUp}$ and $O\mathsf{Encrypt}$.

$|\mathsf{ID}| = Q_{\mathsf{id}}$ is polynomial in this game). This step corresponds to $\mathcal{B}$ betting that $\mathcal{A}$ will choose the client $\mathsf{id}^\star$ in its challenge labels.

When the semantic-security experiment starts, the reduction $\mathcal{B}$ (that is playing as an adversary) gets $\mathsf{ek} = (\mathsf{pk}^\star = sk \cdot P, \mathsf{pp})$ from its challenger $\mathcal{C}$. Then $\mathcal{B}$ starts the data-secrecy experiment (as a challenger) by sending pp to $\mathcal{A}$. The adversary chooses an identity id^* and a pair of keys for it. $\mathcal{A}$ also sends $(\mathsf{id}^*, \mathsf{pk}_{\mathsf{id}^*})$ to $\mathcal{B}$. The reduction registers the (malicious) user id^* in the system $(L_{\mathsf{keys}} \leftarrow L_{\mathsf{keys}} \cup (\mathsf{id}^*, \cdot, \mathsf{pk}_{\mathsf{id}^*}))$ and replies to $\mathcal{A}$'s queries as follows.

sign-up queries: $\mathcal{B}$ forwards the queries to the $O\mathsf{SignUp}$.

encryption queries: $\mathcal{B}$ forwards the queries to the $O\mathsf{Encrypt}$ oracle unless $\ell = (\mathsf{pk}^\star, \mathsf{pk}, \tau)$, in which case $\mathcal{B}$ updates the list of queried labels $L_{\mathsf{lab}} \leftarrow L_{\mathsf{lab}} \cup \ell$, forwards (ℓ, m) as an encryption query to $\mathcal{C}$ and relays its reply to $\mathcal{A}$.

Let (ℓ^*, m_0, m_1) be $\mathcal{A}$'s input to the challenge phase. If $\ell^* \neq (\mathsf{pk}^\star, \cdot, \cdot)$ the reduction aborts (as $\mathcal{A}$ chose to challenge a different client than the one $\mathcal{C}$ has created). This event happens with probability $(1 - \frac{1}{Q_{\mathsf{id}}})$. Otherwise $\ell^* = (\mathsf{pk}^\star, Q, \tau)$, $\mathcal{B}$ updates the list of queried labels $L_{\mathsf{lab}} \leftarrow L_{\mathsf{lab}} \cup \ell^*$ and sends (ℓ^*, m_0, m_1) to its challenger. Let ct denote $\mathcal{C}$'s reply, $\mathcal{B}$ sends ct to $\mathcal{A}$.

In the subsequent query phase $\mathcal{B}$ behaves as described above. At the end of the experiment, $\mathcal{B}$ outputs the same bit b^* returned by $\mathcal{A}$ for the data-secrecy experiment. Note that since $\mathcal{A}$ is given exactly the same challenge as in the semantic-security experiment, if $\mathcal{A}$ has a non-negligible advantage in breaking the data-secrecy of HIKE then $\mathcal{B}$ has the same non-negligible advantage in breaking the semantic security of LEEG, unless $\mathcal{B}$ aborts its simulation. Therefore we can conclude that: $\mathtt{Adv}^{\mathsf{sem.sec}}_{\mathsf{LEEG},\mathcal{B}}(\lambda) \geq \left(\frac{1}{Q_{\mathsf{id}}}\right) \mathtt{Adv}^{\mathsf{data.sec}}_{\mathsf{HIKE},\mathcal{A}}(\lambda)$. □

6.2 Token secrecy

Our notion of token-secrecy captures the idea that only the service provider P holding a valid decryption-token for a ciphertext that was created with P as intended recipient (*i.e.,*with associated label of the form $\ell = (\cdot, \mathsf{pk}_\mathsf{P}, \cdot)$) can decrypt the message correctly. In other words, the adversary $\mathcal{A}$ (as a malicious P^*) should not be able to decrypt the result of any computation $\mathcal{P}^*$ for which it did not received decryption-tokens. We recall that by the token-composability property (Remark 1 in Section 4), given two decryption-tokens $\mathsf{tok}_\mathcal{P}$, $\mathsf{tok}_{\mathcal{P}'}$ for two labelled programs $\mathcal{P}$ and $\mathcal{P}'$, it is possible to generate decryption tokens for any linear combination of the programs $\mathcal{P}$ and $\mathcal{P}'$.

In the token-secrecy experiment, we make use of the same $O\mathsf{SignUp}$ oracle as in the experiment in Figure 4; an $O\mathsf{Encrypt}'$ oracle which is the same as the $O\mathsf{Encrypt}$ in the experiment in Figure 4 except that every time it would output a ciphertext ct it will also add the record to the database, *i.e.,*$\Delta \leftarrow \Delta \cup (\ell, \mathsf{ct})$ where ℓ is the label chosen by $\mathcal{A}$; and an additional $O\mathsf{Disclose}$ oracle, that enables $\mathcal{A}$ to get decryption-tokens of chosen (computations on) records. We allow the adversary to get decryption-tokens for any computation $\mathcal{P}$ as long as this does not contain the challenge labels.

$\underline{\mathtt{Exp}^{\mathsf{token.sec}}_{\mathsf{HIKE},\mathcal{A}}(\lambda)}$:

$b \xleftarrow{\$} \{0,1\}, L_{\mathsf{tok}} = L_{\mathsf{lab}} = L_{\mathsf{keys}} = \emptyset, \Delta = \emptyset$
$\mathsf{pp} \leftarrow \mathsf{Initialise}(1^\lambda)$
$(\mathsf{id}^*, \mathsf{sk}_{\mathsf{id}^*}, \mathsf{pk}_{\mathsf{id}^*}) \leftarrow \mathcal{A}(\mathsf{pp})$
$L_{\mathsf{keys}} \leftarrow L_{\mathsf{keys}} \cup (\mathsf{id}^*, *, \mathsf{pk}_{\mathsf{id}^*})$
$O = \{O\mathsf{SignUp}(\cdot), O\mathsf{Encrypt}'(\cdot,\cdot), O\mathsf{Disclose}(\cdot)\}$
$(\ell^*, m_0, m_1) \leftarrow \mathcal{A}^O(\mathsf{pp})$
let $\ell^* = (\mathsf{pk}_{\mathsf{id}}, \mathsf{pk}_{\mathsf{id}'}, \tau)$
if $\ell^* \notin L_{\mathsf{lab}}$ or $\ell^* \in L_{\mathsf{tok}}$ or $\mathsf{pk}_{\mathsf{id}} = \mathsf{pk}_{\mathsf{id}^*}$
  or $\mathsf{pk}_{\mathsf{id}}, \mathsf{pk}_{\mathsf{id}'} \notin L_{\mathsf{keys}}$
  $\mathsf{ct} = \mathtt{error}$
else
  $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{sk}_{\mathsf{id}}, \ell^*, m_b)$
  $L_{\mathsf{tok}} \leftarrow L_{\mathsf{tok}} \cup \ell^*$, $L_{\mathsf{lab}} \leftarrow L_{\mathsf{lab}} \cup \ell^*$
$b^* \leftarrow \mathcal{A}^O(\mathsf{ct})$
if $b^* = b$ return 1, else return 0.

$O\mathsf{Encrypt}'(\ell, m)$:
parse $\ell = (\mathsf{pk}_{\mathsf{id}}, \mathsf{pk}_{\mathsf{id}'}, \tau)$
 if $\ell \in L_{\mathsf{lab}}$ or $\mathsf{pk}_{\mathsf{id}} = \mathsf{pk}_{\mathsf{id}^*}$
  or $(\cdot,\cdot,\mathsf{pk}_{\mathsf{id}}) \notin L_{\mathsf{keys}}$
  return `error`.
$L_{\mathsf{lab}} \leftarrow L_{\mathsf{lab}} \cup \ell$
$\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{sk}_{\mathsf{id}}, \ell, m)$
$\Delta \leftarrow \Delta \cup (\ell, \mathsf{ct})$
 return ct.

$O\mathsf{Disclose}(\mathcal{P})$:
parse $\mathcal{P} = (f, \ell_1, \ldots, \ell_n)$
with $\ell_i = (\mathsf{pk}_{\mathsf{id}}, \mathsf{pk}_{\mathsf{id}'}, \tau_i)$
 if $\mathsf{pk}_{\mathsf{id}}, \mathsf{pk}_{\mathsf{id}'} \notin L_{\mathsf{keys}}$ or $\mathsf{pk}_{\mathsf{id}} = \mathsf{pk}_{\mathsf{id}^*}$
  or $\ell_i \in L_{\mathsf{tok}}$ for all $i \in [n]$
  return `error`.
$L_{\mathsf{tok}} \leftarrow L_{\mathsf{tok}} \cup \{\ell_1, \ldots, \ell_n\}$
$\mathsf{tok} = \mathsf{AllowAccess}(\mathsf{sk}_{\mathsf{id}}, \mathcal{P})$
 return tok.

Fig. 5: The token-secrecy experiment and the oracles $O\mathsf{Encrypt}'$ and $O\mathsf{Disclose}$

Theorem 2. *The* HIKE *protocol achieves token-security, i.e.,for any PPT adversary $\mathcal{A}$ taking part to the experiment in Figure 5, it holds that:*

$$\mathtt{Adv}^{\mathsf{token.sec}}_{\mathsf{HIKE},\mathcal{A}}(\lambda) = \Pr\left[\mathtt{Exp}^{\mathsf{token.sec}}_{\mathsf{HIKE},\mathcal{A}}(\lambda) = 1\right] - \frac{1}{2} \leq Q_{\mathsf{id}} \cdot \mathtt{Adv}^{\mathsf{sem.sec}}_{\mathsf{LEEG},\mathcal{A}}(\lambda).$$

Due to the space limit, we moved proof of Theorem 2 to Appendix A.

6.3 Forgettability

Our notion of forgettability (forget.sec) captures the idea that after a `forget` request, the target ciphertext does no longer decrypt to the original message. More precisely, there is no way to derive what the original message was from a destroyed ciphertext.

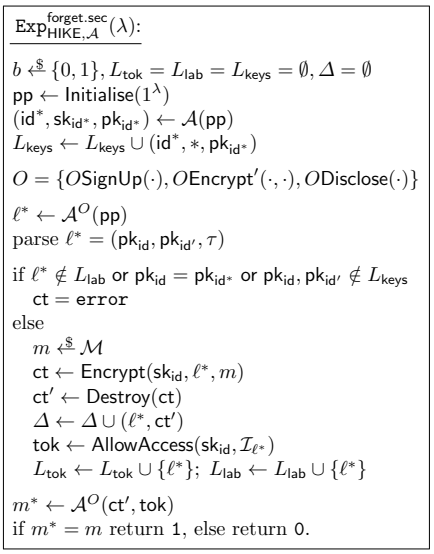


Fig. 6: The forgettability experiment

Our forget.sec experiment, in Figure 6, uses the same oracles as the experiment in Figure 5. Concretely, Experiment 6 is like the token-secrecy experiments (Fig. 5) until the challenge phase. In this phase the forget.sec adversary challenges $\mathcal{C}$ with one single new label ℓ. The challenger then randomly selects a message m, and encrypts it, generates the corresponding decryption token tok for $\mathcal{A}$, and runs the Forget procedure on the challenge ciphertext. Finally $\mathcal{C}$ returns to $\mathcal{A}$ the values $(\mathsf{ct}', \mathsf{tok})$. The adversary's goal is now to correctly guess the challenger's challenge message m. Let m^* denote the output of $\mathcal{A}$ at the end of the experiment in Figure 6, we say that $\mathcal{A}$ wins if $m^* = m$.

It is important to notice that the forget.sec experiment does not model an adversary that is able to obtain the original ciphertext ct, *e.g.,* via a database backup or some previous random `retrieve` request. The main reason for this restriction is the necessity to deploy an access control system on the database Δ which is of independent interest. On the other hand, to avoid an old-ciphertext to be reused, we can only suggest that the client C never distributes the decryption token of queries that involve the label corresponding to *forgotten* ciphertexts.

In a nutshell, our forgettability security statement below says that after a `forget` request the user's record encrypts a random message. In particular, we are able to show that HIKE's Forget procedure achieves information theoretic security in 'hiding' the original message m even under the presence of a malicious server.[5]

Theorem 3. *The* HIKE *protocol achieves perfect forgettability, i.e., for any PPT adversary $\mathcal{A}$ taking part to the experiment in Figure 6, it holds that:*

$$\Pr\left[\mathsf{Exp}^{\mathsf{forget.sec}}_{\mathsf{HIKE},\mathcal{A}}(\lambda) = 1\right] = \frac{1}{|\mathcal{M}|}$$

Proof. The result follows trivially from the information theoretic security of the Destroy algorithm demonstrated in Section 4. □

[5] More precisely, if the server is honest-but-curious except with `forget` requests.

7 Implementation details and results

In this section, we discuss our *encoding map* from the message space to elliptic curve points. Afterwards, we describe the test-settings of our HIKE implementation with respect to different elliptic curve choices.

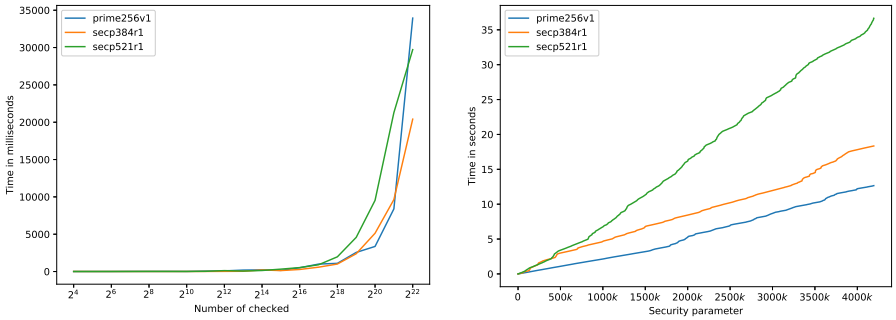
Encoding Messages on the Elliptic Curve. A typical design problem that arises when using *Elliptic Curve Cryptography (ECC)* is to define an injective map ϕ from a message space $\mathcal{M}$ to the subgroup $\mathbb{G}$ generated by a point P on an elliptic curve $\mathcal{E}$. This problem was firstly considered and *"solved"* by Koblitz in [16] by exploiting specific elliptic curves constructed over $\mathbb{F}_{2^n}$ for some appropriate n that depends on the message space dimension.

The main issue with Koblitz's map is that if we equip the message space $\mathcal{M}$ with an operation $\diamond$ and obtain the group $(\mathcal{M}, \diamond)$, then it is generally false that ϕ_K is a homomorphism between $(\mathcal{M}, \diamond)$ and $(\mathbb{G}, +)$, *i.e.,*there exists two messages $m_1, m_2 \in \mathcal{M}$ such that $\phi_K(m_1 \diamond m_2) \neq \phi_K(m_1) + \phi_K(m_2)$. A more natural homomorphism map is given by $\phi : \mathbb{Z}_q \to \mathbb{G}$ as $\phi(m) \coloneqq m \cdot P$. The mapping is trivially a homomorphism when considering the message space as the natural group $(\mathbb{Z}_q, +)$. Unfortunately computing the inverse map ϕ^{-1} is exactly the DLog problem.

In our HIKE protocol we use this natural map to encode messages, and therefore the decryption procedure corresponds to solving an instance of the DLog problem. The apparent contradiction is addressed by the following observation. The security of HIKE relies on the hardness of solving the DLog problem (Assumption 1), but the efficiency of the decryption procedure is guaranteed by the feasibility of solving the IDLP in a particular interval (Assumption 2). In our implementation of HIKE, we consider the natural embedding ϕ and define a context-dependent message-space interval $\mathcal{M} = [a \,...\, b]$, for some $a, b \in \mathbb{N}$. This trick works whenever the decryption knows an approximation of the expected value. This is the case in most of the application scenarios we consider (*e.g.,* range of blood pressure values and range of kilometres run per day). Additionally, the technique does not work when the message space is too big (*e.g.,* floats of 64 bits) or not known.

To demonstrate our claim, we carried out one experiment to test that the decryption algorithm solves the IDLP in a reasonable time (see Figure 7b) whereas a malicious adversary would still face the full DLog problem which is infeasible (see Figure 7a). We implemented an extremely naive brute-force attack that checks, sequentially and incrementally, all the points of the selected interval. For this algorithm the *worst-case* in the interval $[a, \ldots, b]$ is the point $b \cdot P$. In Figure 7a, we empirically measure the running time of the our naive brute-force algorithm to solve the DLog problem with respect to the security parameter. As expected, the problem is exponentially hard. Then, in Figure 7b, we focus on a specific message space, *i.e.,* numbers from 0 to 2^{22} as justified by Assumption 2, and plot the required time needed to decrypt a specific message.

HIKE Implementation. We have developed our HIKE scheme on Python by creating a new cryptographic scheme in the Charm Crypto framework [1]. The



(a) Time needed to solve the DLog. **(b)** Time needed to solve IDLP.

Fig. 7: Comparison between solving the DLog vs solving IDLP

source code of HIKE is freely available at `https://github.com/Pica4x6/HIKE`. For the experiments, we used a MacBook Air with 2,2 GHz Intel Core i7 and 8 GB of RAM. We executed the experiments 100 times independently using the `timeit` library and report the average of the execution times in Table 1.

	KeyGen	Enc	Dec	Eval	PublicKey	TokenGen	TokenDec	Destroy
prime256v1	0.9ms	280.0ms	13442.4ms	20.2ms	5.3ms	1293.2ms	14.5ms	6.4ms
secp384r1	1.0ms	399.7ms	15149.3ms	19.0ms	70.1ms	1521.0.0ms	86.7ms	73.9ms
secp521r1	1.3ms	426.6ms	17102.7ms	189.2ms	66.4ms	1837.5ms	101.2ms	68.0ms

Table 1: Benchmark of HIKE scheme using the natural encoding map ϕ.

In addition, we evaluated the performances of HIKE using the three elliptic curves prime256v1, secp384r1 and secp521r1 that are recommended by the *National Institute of Standard and Technology (NIST)* [18]. Note that our implementation is agnostic to the definition of elliptic curve, thus it can be easily adapted to work with any type of elliptic curves defined in [1].

We remark that for every experiment, we randomly select a message in the HIKE message space dimension in which IDLP is feasible by our Assumption 2 and our empirical test in Figure 7.

8 Conclusions and directions for future work

In this paper, we proposed a new labelled homomorphic encryption scheme for multi-variate linear polynomial functions called LEEG. LEEG can be seen as a variant of ElGamal encryption on elliptic curve groups. We showed that LEEG supports additional features that are not commonly investigated for encryption scheme. We call this set of extra algorithms FEET, as it extends LEEG and improves its versatility. We then combined LEEG and FEET to make HIKE, a lightweight protocol designed for privately and securely store users' data while

keeping it accessible to data owners and authorised service-providers. Application scenarios for HIKE include sport-tracking activity and simple e-Health alter systems. We deployed HIKE on Python and benchmarked its performance. Finally, we included in our security model some GDPR-inspired notions and proved that HIKE provides: *(i)* encrypted storage of the client's data; *(ii)* data owner's right to disclose information (including computation on data) to designated service-providers; and *(iii)* the right to be forgotten, *i.e.,* the possibility for data owners to request that selected records be made un-recoverable.

We identify some direction for further development of our HIKE protocol. First, since HIKE is based on a semantic-secure homomorphic encryption scheme, it cannot tolerate a malicious server. It would be interesting to design protocols with no trust on the server, thus providing both data confidentiality and integrity. Second, there are other extra features (not just FEET) that are worth developing. For example: generation of disclosure-tokens to allow any (chosen) third-party to decrypt a chosen computation on the user's data; introducing a trusted authority (*e.g.,* a legal entity) with the power of decrypting malicious users' data only if it collaborates with the designated service providers; enabling secure *"editable decryption"* to support the *rectification right* (art. 16 in GDPR). Third, it would be worth investigating multi-key properties in LEEG. Such extension would for instance enable service-providers to perform statistic on data generated by different.

To the best of our knowledge, HIKE is the first cryptographic protocol proven to meet specific real-world privacy requirements, and we hope that it constitutes a springing-board for future works. We believe that a GDPR-oriented design of cryptographic protocols and primitives would facilitate developers implementation choices when designing new digital-services, as well as ensure cryptographically-proven security in the data-flow, leading to *privacy-by-design* solutions.

Acknowledgement. We thank the anonymous reviewers for their insightful comments and Erik-Oliver Blass for kindly shepherding us during this publication.

References

1. J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin. Charm: a framework for rapidly prototyping cryptosystems. *Journal of Cryptographic Engineering*, 3(2):111–128, Jun 2013.
2. M. Barbosa, D. Catalano, and D. Fiore. Labeled homomorphic encryption. In *ESORICS*, pages 146–166, 2017.
3. Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. *SIAM Journal on Computing*, 43(2):831–871, 2014.
4. R. Canetti, S. Raghuraman, S. Richelson, and V. Vaikuntanathan. Chosen-ciphertext secure fully homomorphic encryption. In *PKC*, pages 213–240, 2017.
5. A. Connolly. Freedom of encryption. *S&P*, 16(1):102–103, 2018.
6. Council of the European Union, European Parliament. Regulation (EU) 2016/679 (General Data Protection Regulation). `https:`

//publications.europa.eu/en/publication-detail/-/publication/3e485e15-11bd-11e6-ba9a-01aa75ed71a1/language-en, 2016. Online; accessed May 2018.
7. T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, pages 10–18, 1985.
8. D. Fiore, A. Mitrokotsa, L. Nizzardo, and E. Pagnin. Multi-key homomorphic authenticators. In *ASIACRYPT*, pages 499–530. Springer, 2016.
9. A. Fischer, B. Fuhry, F. Kerschbaum, and E. Bodden. Computation on encrypted data using data flow authentication. *CoRR*, abs/1710.00390, 2017.
10. R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, pages 465–482, 2010.
11. R. Gennaro and D. Wichs. Fully homomorphic message authenticators. In K. Sako and P. Sarkar, editors, *ASIACRYPT*, pages 301–320, 2013.
12. C. Gentry. Fully homomorphic encryption using ideal lattices. pages 169–178, 2009.
13. S. Goldwasser and S. Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *STOC*, pages 365–377. ACM, 1982.
14. J. Hoffstein, J. Pipher, J. H. Silverman, and J. H. Silverman. *An introduction to mathematical cryptography*, volume 1. Springer, 2008.
15. J. Katz and Y. Lindell. *Introduction to modern cryptography*. CRC press, 2014.
16. N. Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.
17. R. Montenegro and P. Tetali. How Long Does It Take to Catch a Wild Kangaroo? In *STOC*, pages 553–560. ACM, 2009.
18. NIST STS. Cryptographic Key Length Recommendation. https://www.keylength.com/en/4/, 2017. Online; accessed May 2018.
19. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In J. Stern, editor, *EUROCRYPT*, pages 223–238, 1999.
20. M. Panjwani and M. Jäntti. Data protection & security challenges in digital & it services: A case study. In *ICCA*, pages 379–383. IEEE, 2017.
21. J. M. Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of computation*, 32(143):918–924, 1978.
22. J. M. Pollard. Kangaroos, Monopoly and Discrete Logarithms. *Journal of Cryptology*, 13(4):437–447, Sept. 2000.
23. R. L. Rivest, L. Adleman, and M. L. Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation, Academia Press*, 1978.
24. N. P. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *PKC*, pages 420–443, 2010.
25. M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT*, pages 24–43. Springer, 2010.

Appendix

A Detailed proofs

Semantic security of LEEG. We want to prove that our LEEG scheme is semantically secure according to Barbosa *et al.*'s definition [2] (Definition 4).

Proof. Let $Q_{\mathsf{prf}}(\lambda)$ be a bound on the total number of encryption queries performed by $\mathcal{A}$ during the security experiment. Let Game 0 be the semantic security experiment in Definition 4 where, for consistency with our definition of LEEG, the challenger runs $\mathsf{SetUp}(1^\lambda) \to \mathsf{pp}$ to obtain the public parameters of the scheme, then it runs $\mathsf{KeyGen}(\mathsf{pp}) \to \mathsf{sk} = (sk, \mathsf{k})$ and computes $\mathsf{pk} = sk \cdot P$. In addition, $\mathcal{C}$ ignores any query with label $\ell = (Q', Q, \tau)$ where $Q' \neq \mathsf{pk}$.

Let Game 1 be the same as Game 0 except that the challenger replaces every $\mathsf{PRF}_{\mathsf{k}}(\cdot)$ instance with the evaluation of a truly random function $\mathsf{rand} : \mathbb{G} \times \mathbb{G} \times \mathcal{T} \to [0..q-1]$. It is quite easy to see that the difference between Game 1 and Game 0 is solely in the generation of the values r. Therefore the probability of $\mathcal{A}$ winning is the same in the two games, a part from a $Q_{\mathsf{prf}}(\lambda)$ factor that comes from distinguishing the PRF instance from a truly random function. Thus

$$|\mathsf{Prob}[\mathsf{G}_0(\mathcal{A})] - \mathsf{Prob}[\mathsf{G}_1(\mathcal{A})]| \leq Q_{\mathsf{prf}}(\lambda) \cdot \mathtt{Adv}^{\mathsf{PRF}}_{\mathcal{A}}(\lambda).$$

At this point, we observe that for any given ciphertext ct and label-message pair (ℓ, m) there is exactly one value $r \in [0..q-1]$ for which ct is an encryption of m for label ℓ. In particular, for every triple (ct, ℓ, m) it holds that

$$\mathsf{Prob}[\mathsf{Enc}(\mathsf{sk}, \ell, m) = \mathsf{ct}] = \frac{|r \in [0..q-1] : M + r \cdot sk \cdot Q = \mathsf{ct}|}{|[0..q-1]|} = \frac{1}{q},$$

where the probability is taken over all the possible values $r \xleftarrow{\$} [0..q-1]$. Since the above probability holds also for the challenge ciphertext ct^* we have that $\mathsf{Prob}[\mathsf{Enc}(\mathsf{sk}, \ell_0, m_0) = \mathsf{ct}^*] = \mathsf{Prob}[\mathsf{Enc}(\mathsf{sk}, \ell_1, m_1) = \mathsf{ct}^*]$ (semantic security) and implies $\mathsf{Prob}[\mathsf{G}_1(\mathcal{A})] = \frac{1}{2}$. Therefore:

$$\begin{aligned}\mathsf{Prob}[\mathsf{G}_0(\mathcal{A})] &\leq |\mathsf{Prob}[\mathsf{G}_0(\mathcal{A})] - \mathsf{Prob}[\mathsf{G}_1(\mathcal{A})]| + \mathsf{Prob}[\mathsf{G}_1(\mathcal{A})] \\ &\leq Q_{\mathsf{prf}}(\lambda) \cdot \mathtt{Adv}^{\mathsf{PRF}}_{\mathcal{A}}(\lambda) + \frac{1}{2}\end{aligned}$$

which proves the semantic security of LEEG, given that $Q_{\mathsf{prf}}(\lambda)$ is polynomial and $\mathtt{Adv}^{\mathsf{PRF}}_{\mathcal{A}}$ is negligible (by our security assumption on the PRF family). □

Proof of Theorem 2. We want to prove our HIKE protocol achieves token secrecy, *i.e.,* that $\mathtt{Adv}^{\mathsf{token.sec}}_{\mathsf{HIKE},\mathcal{A}}(\lambda) \leq Q_{\mathsf{id}} \cdot \mathtt{Adv}^{\mathsf{sem.sec}}_{\mathsf{LEEG},\mathcal{A}}(\lambda)$.

Proof. We exhibit a reduction $\mathcal{B}$ that uses $\mathcal{A}$ to win the semantic-security experiment for the LEEG scheme. The reduction works exactly as the one in the proof

of Theorem 1 a part for a couple of exceptions. First, this reduction holds an additional (private) list L_{rand}, that is empty at the beginning of the simulation. Second, $\mathcal{B}$ behaves differently (only) in the following cases:

Encryption queries: $\mathcal{B}$ forwards the queries to the $O\mathsf{Encrypt}'$ oracle, unless $\ell = (\mathsf{pk}^\star, \mathsf{pk}, \tau)$. In case $\ell = (\mathsf{pk}^\star, \mathsf{pk}, \tau)$, the reduction does not have the secret key for encryption and token generation. In order to simulate the encryption and be consistent with future token-generation queries, $\mathcal{B}$ checks if $(\ell, r) \in L_{\mathsf{rand}}$ for some value $r \in [0..q-1]$. If so, $\mathcal{B}$ uses the existing values r to compute the ciphertext $\mathsf{ct} = (m \cdot P + r \cdot \mathsf{pk})$. Otherwise, $\mathcal{B}$ picks a random value $r \xleftarrow{\$} [0..q-1]$, updates $L_{\mathsf{rand}} \leftarrow L_{\mathsf{rand}} \cup (\ell, r)$, and computes $\mathsf{ct} = (m \cdot P + r \cdot \mathsf{pk})$. In any case, $\mathcal{B}$ updates the list of queried labels $L_{\mathsf{lab}} \leftarrow L_{\mathsf{lab}} \cup \ell$, and returns ct to $\mathcal{A}$. Note that ct has the same distribution as the output of $\mathsf{Enc}(\mathsf{sk}^\star, \ell, m)$, indeed for any r chosen by the reduction there exists a value $r' \in [0..q-1]$ such that $r = r' \cdot sk^\star \mod q$ and thus $\mathsf{ct} = m \cdot P + r \cdot \mathsf{pk} = m \cdot P + r' \cdot \mathsf{sk}^\star \cdot \mathsf{pk}$. The latter series of equalities shows that $\mathcal{B}$'s simulation is still perfect.

Disclose queries: $\mathcal{B}$ forwards to the $O\mathsf{Disclose}$ oracle all the queries $\mathcal{P} = (f, \ell_1 \ldots, \ell_n)$ with $f(x_1, \ldots, x_n) = a_0 + \sum_{i\in[n]} a_i x_i$, $\ell_i = (\mathsf{pk}, \mathsf{pk}', \tau_i)$ and $\mathsf{pk} \neq \mathsf{pk}^\star$. Otherwise, $\mathcal{P}$ contains labels of the form $\ell_i = (\mathsf{pk}^\star, \mathsf{pk}', \tau_i)$. The reduction performs the same checks as the $O\mathsf{Disclose}$ oracle, if any check fails $\mathcal{B}$ returns `error`. In case all conditions are met, $\mathcal{B}$ proceeds by checking if $(\ell_i, \cdot) \in L_{\mathsf{rand}}$ for all $i \in [n]$, in which case the reduction uses the randomness stored in L_{rand} to compute the token $\mathsf{tok} = (\sum_{i\in[n]} a_i r_i) \cdot \mathsf{pk}^\star$. Otherwise, for all those labels ℓ_j not present in L_{rand}, $\mathcal{B}$ samples a random element $r \xleftarrow{\$} [0..q-1]$ and updates the private list $L_{\mathsf{rand}} \leftarrow L_{\mathsf{rand}} \cup (\ell_j, r)$. At this point $(\ell_i, r_i) \in L_{\mathsf{rand}}$ for all the labels in the queried $\mathcal{P}$ and $\mathcal{B}$ can compute $\mathsf{tok} = (\sum_{i\in[n]} a_i r_i) \cdot \mathsf{pk}^\star$. In either case, $\mathcal{B}$ updates the list of queried token-labels, *i.e.,* $L_{\mathsf{tok}} \leftarrow L_{\mathsf{tok}} \cup (\ell_1, \ldots, \ell_n)$, and returns tok to $\mathcal{A}$.

Let (ℓ^*, m_0, m_1) be $\mathcal{A}$'s input to the challenge phase. If $\ell^* \neq (\mathsf{pk}^\star, \cdot, \cdot)$ the reduction aborts ($\mathcal{A}$ chose to challenge a different client than the one $\mathcal{B}$ bet on). This event happens with probability $(1 - \frac{1}{Q_{\mathsf{id}}})$.

Otherwise $\ell^* = (\mathsf{pk}^\star, Q, \tau)$, $\mathcal{B}$ updates the list of queried labels $L_{\mathsf{lab}} \leftarrow L_{\mathsf{lab}} \cup \ell^*$ and sends (ℓ^*, m_0, m_1) to its challenger $\mathcal{C}$ for the semantic security game. Let ct denote $\mathcal{C}$'s reply, $\mathcal{B}$ forwards ct to $\mathcal{A}$.

In the subsequent query phase $\mathcal{B}$ behaves as described above.

At the end of the experiment, $\mathcal{B}$ outputs the same bit b^* returned by $\mathcal{A}$ for the $\mathsf{token.sec}$ experiment. Note that since $\mathcal{A}$ is given exactly the same challenge as in the $\mathsf{sem.sec}$ experiment, if $\mathcal{A}$ has a non-negligible advantage in winning the $\mathsf{token.sec}$ experiment, then $\mathcal{B}$ has the same non-negligible advantage in breaking the semantic-security of LEEG, unless $\mathcal{B}$ aborts its simulation. Therefore we conclude that: $\mathtt{Adv}^{\mathsf{sem.sec}}_{\mathsf{LEEG},\mathcal{B}}(\lambda) \geq \left(\frac{1}{Q_{\mathsf{id}}}\right) \cdot \mathtt{Adv}^{\mathsf{token.sec}}_{\mathsf{HIKE},\mathcal{A}}(\lambda)$. □

B Token composability in FEET

In this appendix we show token composability for two labelled programs. The general case follows immediately.

Consider two labelled programs $\mathcal{P}_1 = (f_1, \ell_1, \dots, \ell_n)$ and $\mathcal{P}_2 = (f_2, \ell'_1, \dots, \ell'_{n'})$. For consistency, token composability requires that all the labels involved in $\mathcal{P}_1$ and $\mathcal{P}_2$ are of the form $\ell = (sk \cdot P, Q, \tau)$ for some opportune value of τ. Without loss of generality, we can set $f_1 = a_0 + \sum_{i\in[n]} a_i x_i$ and $f_2 = a'_0 + \sum_{j\in[n']} a'_j x_{\sigma(j)}$ for opportune coefficients $a_i, a'_j \in \mathcal{M}$, and an index-mapping function $\sigma : \mathbb{N} \to \mathbb{N}$ used to model the fact that the functions may be defined on a different set of variables. Let $I \subseteq \mathbb{N}$ be the set of indexes of common variables, formally:

$$I = \{i \in [n] \text{ such that } \sigma(j) = i \text{ for some } j \in [n']\}.$$

The composed labelled program $\mathcal{P} = b_1\mathcal{P}_1 + b_2\mathcal{P}_2$ is defined as $\mathcal{P} = (f, \tilde{\ell}_1, \dots, \tilde{\ell}_{\tilde{n}})$ with $f = b_1 f_1 + b_2 f_2$, $f(x_1, \dots, x_{\tilde{n}}) = (b_1 a_0 + b_2 a'_0) + \sum_{i\in I}(b_1 a_i + b_2 a'_i)x_i + \sum_{i\in[n]\setminus I} b_1 a_i x_i + \sum_{j\in[n']\setminus\sigma(I)} b_2 a'_j x_j$ for any $b_1, b_2 \in \mathcal{M}$. We show that the combined token $\mathsf{tok} = b_1\mathsf{tok}_1 + b_2\mathsf{tok}_2$ is a valid decryption token for the composed labelled program $\mathcal{P}$, actually $\mathsf{tok} = \mathsf{TokenGen}(\mathsf{sk}, \mathcal{P})$. In details:

$$\begin{aligned}
\mathsf{tok} &= b_1\mathsf{tok}_1 + b_2\mathsf{tok}_2 \\
&= sk \cdot \Big(\sum_{i\in[n]} b_1 a_i r_i + \sum_{j\in[n']} b_2 a'_j r'_{\sigma(j)} \Big) \cdot P \\
&= sk \cdot \Big(\sum_{i\in I} (b_1 a_i + b_2 a'_i) r_i + \sum_{i\in[n]\setminus I} b_1 a_i r_i + \sum_{j\in[n']\setminus\sigma(I)} b_2 a'_j r'_j \Big) \cdot P \\
&= \mathsf{TokenGen}(\mathsf{sk}, \mathcal{P})
\end{aligned}$$

The set I in the second last equality is the one defined in (5), that is $\ell_i = \ell'_{i=\sigma(j)}$ for all $i \in I$ and therefore $r_i = r'_i = \mathsf{PRF}_\mathsf{k}(\ell_i)$. By the correctness of the TokenGen-TokenDec algorithms, we derive that tok is a valid decryption token for sk_2, $\mathsf{ct} = \mathsf{Eval}(f, \mathsf{ct}_1, \dots, \mathsf{ct}_{\tilde{n}})$. It is straightforward to generalise this reasoning to multiple labelled programs $\mathcal{P}_1, \dots \mathcal{P}_t$ as long as all the labels coincide on the first two entries.