Labeled Homomorphic Encryption: Scalable and Privacy-Preserving Processing of Outsourced Data

Manuel Barbosa[1], Dario Catalano[2], and Dario Fiore[3]

[1] HASLab – INESC TEC DCC FC University of Porto, Portugal
[2] University of Catania Italy
[3] IMDEA Software Institute Madrid, Spain

Abstract. We consider the problem of privacy-preserving processing of outsourced data, where a Cloud server stores data provided by one or multiple data providers and then is asked to compute several functions over it. We propose an efficient methodology that solves this problem with the guarantee that a honest-but-curious Cloud learns no information about the data and the receiver learns nothing more than the results. Our main contribution is the proposal and efficient instantiation of a new cryptographic primitive called *Labeled Homomorphic Encryption* (labHE). The fundamental insight underlying this new primitive is that homomorphic computation can be significantly accelerated whenever the program that is being computed over the encrypted data is known to the decrypter and is not secret—previous approaches to homomorphic encryption do not allow for such a trade-off. Our realization and implementation of labHE targets computations that can be described by degree-two multivariate polynomials, which capture an important range of statistical functions. As a specific application, we consider the problem of privacy preserving Genetic Association Studies (GAS), which require computing risk estimates for given traits from statistically relevant features in the human genome. Our approach allows performing GAS efficiently, non interactively and without compromising neither the privacy of patients nor potential intellectual property that test laboratories may want to protect.

1 Introduction

Privacy-preserving data processing techniques are crucial enablers for moving many security-critical applications to the Cloud, and they may be the key to unlocking new socially-relevant applications and business opportunities. As an example, consider the case of personalized medicine, where a medical center offers highly specialized services that permit guiding the medical care of a Client based on information encoded in the Genome. Such direct-to-consumer services are already a reality, so we will not discuss whether or not they are desirable. Instead, we propose a new methodology that can be used *today* to deploy such services in the Cloud (genomic studies may involve a huge amount of data), whilst protecting the privacy of the Client, and intellectual property that may be a concern for the medical center. Controlling who has access to individual data in these scenarios will likely be mandatory for ethical and/or legal reasons, and this pattern arises in many other real-world applications (e.g., analysis of taxpayers' or consumers' data, users' geographic locations, etc.) where our solution may be of use.

Fig. 1. The parties and workflow of our system.

We consider a scenario with three actors – data providers, the Cloud, and a receiver – with the following workflow (Fig. 1). Data providers send data to the Cloud, and the receiver asks the Cloud to execute certain queries on the outsourced data. For the applications we consider, the key requirements are *privacy* and *efficiency*. Privacy properties should guarantee that the

Cloud does not learn any information on the hosted data, and that the receiver learns nothing more than the queries outcomes. Furthermore, it should be possible for many data providers to contribute with inputs to the same computation, in such a way that data introduced by one provider is protected from the others. The efficiency requirement involves two main aspects: *computation* and *communication*. With respect to computation, the protocol should have minimal impact for data providers. There is little point for them in delegating storage and/or computation to the Cloud if this requires prohibitive costs; their only task should be to collect and send data and be minimally involved in the rest of the protocol (e.g., they could go offline). Moreover, in several applications the data providers can be resource-constrained devices (e.g., sensors) for which a lightweight protocol is essential. In terms of computation, the protocol should also run efficiently at the Cloud. Although Cloud providers have powerful resources, in an outsourcing setting one has to pay for them and thus the lighter is the protocol's burden the cheaper is the service's cost. On the communication side, one would like solutions with minimal bandwidth overhead both between data providers and the Cloud, and between the Cloud and the receiver. For example, the communication with the receiver should not depend on the amount of data hosted by the Cloud. Low bandwidth is particularly relevant in the context of mobile networks and mobile devices: high bandwidth consumptions drain batteries and cost a lot due to the price of mobile network connections (most of the times under a pay-per-use model).

Our Contribution. We propose and efficiently instantiate a new cryptographic primitive called *Labeled Homomorphic Encryption* (labHE) that gives a solution to the problem of privately processing outsourced data outlined above. Our realization and implementation of labHE targets computations that can be described by degree-two multivariate polynomials, which capture a significant fraction of statistical functions and, in particular, statistical computations used in genomic analysis. As we detail later, our solution outperforms protocols based on previous somewhat homomorphic encryption schemes in essentially all fronts: our communication costs are more than two orders of magnitude smaller, computation is more than 80 times faster for data providers and up to 9000 times faster for the Cloud. The insight that unlocks such performance gains is that homomorphic computation can be significantly accellerated whenever the program that is being computed over the encrypted data is known to decrypter and is not secret—previous approaches to homomorphic encryption do not allow for such a trade-off.

1.1 An Overview of Our Results

Labeled Homomorphic Encryption. Our new labHE notion combines the model of *labeled programs*, put forward in the context of homomorphic authenticators (e.g. [20,10,12]), with the concept of homomorphic encryption. Homomorphic encryption (HE) [36,21] is like ordinary encryption with the additional capability of a (publicly executable) evaluation algorithm `Eval`. The latter takes as input a program P and encrypted messages $m_1, \ldots, m_n$, and outputs an encryption of $P(m_1, \ldots, m_n)$.

labHE is similar to HE with the following additions. First, every piece of (encrypted) data is associated with a unique label. A label could be the index of a database record or any other string that can be used to identify the outsourced data item. Thus, when encrypting a message m, one specifies a corresponding label τ (which does not need to be kept secret, though). To give an example, think of a blood pressure sensor which collects measurements at regular time instants: the pressure value is the actual data while the time instant is the label. Next, whenever a user Bob wants to ask the cloud to compute f on some (previously outsourced) encrypted inputs, he makes

the query by specifying the labels of these inputs. For instance, Bob may say *"compute the mean on messages with labels* $(Pressure, 1), \ldots, (Pressure, 100)$*"*. The combination of f and the labels in the query is called a "labeled program" P, which is what is executed by the Cloud. Finally, upon the receipt of the (encrypted) answer c from the Cloud, Bob runs the decryption algorithm with his secret key, c, and labeled program P. Introducing labeled programs in HE formalizes the intuition that Bob is decrypting the result of a known function (the labeled program, the query) on the unknown outsourced data (the encrypted messages). We stress that in the outsourcing setting labeling is always implicit, as some mechanism is always needed to specify the portion of the outsourced data over which the Cloud has to compute. Moreover, although one may wonder that labels leak additional information, it is not hard to see that this can be avoided by choosing an appropriate labeling (e.g., simple indices) which reveals only trivial information.

For efficiency we require labHE ciphertexts to be *succinct*, i.e., of some fixed size, independent of the computation possibly executed on it. We concede that the running time of labHE decryption may depend on P: this is the most noticeable difference with standard HE. Interestingly, however, in our realizations such dependence has almost negligible impact on efficiency in practice.

For security, we require labHE to meet the usual semantic security notion (i.e., one cannot tell apart encryptions of known messages) and also to satisfy a property that we call *context-hiding*. This essentially says that a ciphertext encrypting the result $m = P(m_1, \ldots, m_n)$ reveals only m and nothing more about the program inputs.

Basic and Multi-user labHE. The basic labHE notion requires the same secret key to encrypt and decrypt. It can be used to perform privacy-preserving computations on outsourced data as follows. A data provider, Alice, jointly executes the setup algorithm with Bob, the receiver, and gets a secret encryption key that she can use to encrypt her data before outsourcing it to the Cloud. Bob can then ask the Cloud to compute a labeled program P on Alice's data, obtain an encryption of the result and decrypt this with his secret decryption key. In terms of data privacy, labHE semantic security ensures that, as long as the Cloud does not get to see the keys used for encryption/decryption, it does not learn *anything* about Alice's data or the result of the computation; context-hiding further guarantees that, as long as the Cloud does not reveal the originally encrypted ciphertexts to Bob, then Bob learns only the query results and no other information about Alice's individual data. We note that this trust model is particularly well suited to a scenario in which Alice (or more of the senders in the multi-sender scenario below) controls the Cloud and uses it to offer a service to Bob. Regarding efficiency, the only work of Alice is to encrypt and transmit the data, while the succinctness of labHE yields short communication between the Cloud and Bob: answers received by Bob do not depend on the size of the outsourced data.

In addition to basic labHE, we also provide a more powerful generalization to a multi-user setting, which inherits all the performance features of the basic one. Here one can perform computations over data encrypted by different providers, and these do not need to share any common secret. Indeed, key generation in the basic labHE notion can be split between sender and receiver as follows. Bob generates a master public key and a master secret key. Knowing Bob's master public key, Alice can unilaterally encrypt with her own generated encryption key, and create a public key that becomes associated with her encrypted data. In this way, no trusted a priori set-up is required in addition to a PKI. Moreover, multiple senders can do exactly the same as Alice to encrypt under their public keys and Bob's master public key, with the extra guarantee that the data encrypted by one sender cannot be decrypted by a different sender. Decryption requires knowledge of the master secret along with the public keys of all the users whose ciphertexts were involved in the computation.

On the usefulness of labeling programs. The essence of labHE is to take advantage of the fact that, when delegating some computation P on outsourced data, P is typically provided explicitly to the cloud. Interestingly, when using (standard) homomorphic encryption this inherent privacy loss does not seem to be exploitable to gain efficiency. labHE, on the other hand, aims at trading the (unavoidable!) leak of P to *significantly* reduce the cost of the computation.

Indeed, the main difference with respect to (standard) homomorphic encryption is in decryption: decrypting in labHE requires Bob to do work that depends on the program P. More precisely, and simplifying things a bit, Bob will basically need to recompute P on (values related to) the *labels* corresponding to the original inputs. Interestingly we show that, as this computation is performed on *unencrypted* and very succinct data (short pseudorandom fingerprints of the labels), it has very low impact in practice. In fact, the cost of decryption is always *orders* of magnitude lower than that of running the computation in the Cloud. Not only that, *this can be done prior to receiving the encrypted results from the Cloud*! This becomes particularly interesting when considering that our realizations of labHE are extremely efficient *also* for the Cloud (see below for more details about this). Indeed, we show that, building on [11], labHE supporting computations expressible via degree-2 polynomials can be realized from any encryption scheme that is only *linearly* homomorphic. Since these are typically more efficient than their more expressive counterparts, the same holds for the resulting labHE.

To the best of our knowledge, the idea of trading-off function privacy for efficiency has not been previously applied in the field of (somewhat) homomorphic encryption; for this reason, and while our work focuses on the specific case of computing degree two polynomials on ciphertexts, we believe that this idea could be of independent interest and might find applications for settings requiring more expressive computations as well.

An Overview of Our Techniques. We provide an intuitive description of our solution, discussing some of the core ideas underlying it. We encrypt a message $m \in \mathcal{M}$ via a two-component ciphertext $(m - b, \mathsf{Enc}(b))$, where Enc is a linearly homomorphic encryption scheme and b is random in $\mathcal{M}$. In [11], Catalano and Fiore show that ciphertexts of this form allow for the evaluation of degree-two polynomials on encrypted data, at the cost of losing compactness. More precisely, Catalano and Fiore argue that when applying a polynomial f on $(m_1 - b_1, \mathsf{Enc}(b_1)), \ldots, (m_t - b_t, \mathsf{Enc}(b_t))$, there may be the possibility (depending on the structure of f) to end up with a huge $O(t)$-components ciphertext $(\mathsf{Enc}(f(m_1, \ldots, m_t) - f(b_1, \ldots, b_t)), \mathsf{Enc}(b_1), \ldots, \mathsf{Enc}(b_t))$.

Our key idea to solve the compactness issue in the context of labHE is to let every b_i depend on the corresponding label; in our construction we set b_i as the output of $F_K(\tau_i)$, where F is a pseudorandom function and τ_i is the unique label associated with message m_i. The crucial observation is that, because the labels are known to the decryptor, the value $f(b_1, \ldots, b_t)$ can be reconstructed at decryption time, and the components $\mathsf{Enc}(b_1), \ldots, \mathsf{Enc}(b_t)$ dismissed from the above ciphertext. This gives us a construction that supports all degree-two polynomials with constant-size ciphertexts!

Interestingly, this simple idea, when instantiated with fast cryptographic primitives (e.g., the Sponge-based pseudorandom function from the Kekkac Code Package and the Joye-Libert cryptosystem [28] as linearly-homomorphic encryption) yields an extremely efficient realization of the primitive, that allows to outsource the computation of various useful functions (e.g. statistics, genetic association studies) in a very efficient yet privacy preserving way.

Efficient labHE Realizations. We show how to construct expressive labHE schemes for quadratic functions by using standard number theoretic (linearly-homomorphic) encryption schemes, such as Paillier [34], Bresson *et al.* [9] and Joye-Libert [28]. We implemented one of these instantiations – the

one based on the Joye-Libert cryptosystem that we call labHE(JL13) – and tested its performance for the case of computing statistical functions on encrypted data. Our experiments demonstrate that labHE(JL13) outperforms a solution based on state-of-the-art somewhat homomorphic encryption (FV) [18,33] (optimized to support the same class of functions) on essentially all fronts. For example, comparing labHE(JL13) against FV, we observed that in labHE(JL13) the communication costs are 400 times smaller, encrypting is more than 80 times faster, while computing the results is between 9000 and 50 times faster for the Cloud.

1.2 Applications

To further highlight the performance benefits of our solution in the real world, we looked at two specific applications: i. computing relevant statistical functions over encrypted data outsourced to the Cloud and ii. peforming Genetic Association Studies that preserve both the privacy of users and the intellectual property of the laboratories performing the tests.

Outsourcing Privacy Preserving Statistics. We consider the case where a potentially large dataset is stored on an (untrusted) Cloud. The latter is used both to store and to perform computations on encrypted data on behalf of one (or more) Clients. More precisely we considered two scenarios. One where the Client acts both as Data Provider and Receiver and a three party scenario where these roles are played by different users/entities. Of course, a solution to the problem of computing secure statistics in these scenarios can be obtained via somewhat homomorphic encryption schemes supporting quadratic polynomials. labHE, however, achieves the same goal with unprecedented efficiency both in terms of computation costs and in terms of bandwidth consumption. In our experiments, we considered multidimensional datasets represented as $(n \times d)$ matrices $X = \{x_{i,j}\}$, where n are the dataset members and d the dimension (or number of variables). Univariate statistics such as Mean and Variance are computed column-wise (e.g., the mean of the j-th column is $\mu_j = \frac{1}{n}\sum_{i=1}^{n} x_{i,j}$), whereas bivariate correlation ones such as Covariance act over pairs of columns. In this setting, if we consider a dataset of over two million entries ($n = 2^{20} \times d = 2$), the solution based on the FV somewhat homomorphic encryption requires over 249GB of storage at the Cloud whereas labHE(JL13) only 560Mbytes. Moreover, for such large datasets the amount of memory required to perform homomorphic computations using FV placed it out of reach of the standard machines we used for benchmarking (scalability is bounded at around 30K elements for 16GB of RAM) while labHE(JL13) scaled up easily to two million entries. When considering the more modest datasets (where FV could run) the cumulative time of computing a Covariance matrix on the encrypted dataset and decrypting its result is 32 minutes using FV and 37 seconds with labHE(JL13); computing and decrypting a Mean query takes about 9 seconds with FV and around 19 milliseconds with labHE(JL13).

Privacy Preserving GAS. Genetic Association Studies (GAS) look for statistically relevant features across the human genome. The goal is to single out those that can be correlated to given traits. Typically such studies are carried by performing series of tests. Each test targets a particular trait and takes into consideration associated information that is encoded in specific positions of an individual's genome, the so-called Single Nucleotide Polymorphisms (SNP). Each test computes a Genetic Risk Score: a weighted sum of the information collected for each SNP and the weights correspond to risk estimates computed for a reference population. This SNP genotyping has already several applications, ranging from personalized medicine to forensics.

Access to such tests is, for the most part, controlled by the health services of different countries, but a new trend of Direct-to-Consumer (DTC) genomic analysis is arising, where companies offer a multitude of association tests to the public. Privacy is obviously a paramount concern in such services. Our (multi-user) Labeled Homomorphic Encryption technology allows to perform GAS securely and without compromising the privacy of the patient. Moreover, it offers a trade-off between security and efficiency that makes it an ideal candidate for practical deployment. Our solution also protects potential intellectual property (e.g., the weights above) that the laboratories carrying out the tests may want to protect, and it may be used in a commodity cloud, since all computation is carried out over encrypted data.

To evaluate the scalability of our solution we considered a Map-Reduce scenario where the multiplicative part of the weighted sum is split by multiple servers in the Cloud. In this way many partial sums can be computed in parallel and later combined to get the final result. Using this strategy, a GAS computation including over 1 million SNPs can be completed in roughly 3 minutes using 10 servers (excluding communication overhead). Using FV [18], as underlying building block for a risk analysis involving only 30K SNPs the size of encrypted data processed by the Cloud becomes, roughly, 14 GBytes, which is over 400 times more than the space required by our solution. For the same task, FV-based solutions turn out to be around 100 times slower than our solution. This comparison is for a modest number of SNPs since, for larger parameters, experiments became highly unstable and eventually infeasible due to too large memory requirements that surpassed the capabilities of our benchmarking platform.

Solutions based on Related Primitives. A solution to the problem considered in this paper can also be obtained via the use of fully homomorphic encryption (FHE) [21]. Although fully fledged constructions of FHE are too inefficient to be used in practice, less powerful variants (often referred to as *somewhat homomorphic encryption* – SHE) are more efficient. Still, even when restricting to very specific functionalities (e.g., degree-2 polynomials), known somewhat homomorphic encryption schemes are less efficient than our solution, as we mentioned in the previous paragraphs. One reason is that the most efficient SHE constructions are based on the hardness of lattice problems, and to achieve meaningful security they require parameters that induce very large ciphertexts. Another aspect to take into consideration is that lattice-based cryptosystems lack of standardized parameters, and are harder to implement for the less expert practitioners (while there are some libraries for lattice based homomorphic encryption [24,33], these are way less developed and popular than existing ones for widely standardized number theoretic cryptography). Also, we remark that FHE-based solutions would allow secure outsourcing in the same colluding model (i.e. security is guaranteed only as long as the receiver and the Cloud do not collude).

Another solution to the outsourcing problem can come from the field of multiparty computation (MPC). In particular, there is a category of MPC protocols (e.g., [5,16,15]) in which participants can be divided into input parties, computing parties (e.g., several Cloud servers) and result parties. These protocols (partially) fit the outsourcing scenario addressed in this paper: computing parties who perform the actual computation are distributed across *multiple* (often interacting) Cloud servers. Notably, these servers should be deployed by separate organizations that are assumed not to collude. This is however an assumption which induces further complexities in practice, e.g., one should make sure to use cloud companies in separate geographic locations under different legal jurisdictions, deal with legal regulations in each of these countries, etc. In contrast, our solution works in a much simpler setting where one relies on a single, untrusted, Cloud.

Finally, we mention that a trivial solution for computing on outsourced encrypted data is the following: a user Alice stores her data in encrypted format on the Cloud; whenever Bob wants to perform computations on Alice's data, he simply downloads (and decrypts) the relevant ciphertexts locally, and performs the computation on his own. We call this solution "download&decrypt" (D&D). "Download&decrypt" may fit applications where the same user uploads the data and later decrypts the results (i.e., Alice and Bob are the same entity). However, D&D is *not* a solution for our scenario because it does not fit our privacy requirements, since Bob learns everything about Alice's data. This is in contrast to our solution based on labHE where context-hiding guarantees that, as long as Bob and the Cloud do not collude, Alice's data remains private to Bob (except for what he is intended to learn honestly from the queries outputs). On top of this consideration, we mention an additional reason by which our solution compares favorably against D&D. In our case Bob pays communication that depends only on the size of the results. In D&D, Bob should either download and store the entire input, or request each input every time he needs it in the computation. If Bob has limited storage space or he is paying for the consumed bandwidth, D&D becomes unaffordable for large datasets.

2 Preliminaries

Notation. We denote with $\lambda \in \mathbb{N}$ a security parameter, and with $\mathsf{poly}(\lambda)$ any function bounded by a polynomial in λ. We say that a function ϵ is *negligible* if it vanishes faster than the inverse of any polynomial in λ. We use PPT for probabilistic polynomial time, i.e., $\mathsf{poly}(\lambda)$. If S is a set, $x \xleftarrow{\$} S$ denotes selecting x uniformly at random in S. If $\mathcal{A}$ is a probabilistic algorithm, $x \xleftarrow{\$} \mathcal{A}(\cdot)$ denotes the process of running $\mathcal{A}$ on some appropriate input and assigning its output to x. For a positive integer n, we denote by $[n]$ the set $\{1, \ldots, n\}$.

Definition 1 (Statistical Distance). *Let X, Y be two random variables over a finite set $\mathcal{U}$. We define the statistical distance between X and Y as*

$$\mathsf{SD}[X, Y] = \frac{1}{2} \sum_{u \in U} \left| \Pr[X = u] - \Pr[Y = u] \right|$$

2.1 Homomorphic Encryption

A homomorphic encryption scheme HE is a tuple of four PPT algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ as follows:

$\mathsf{KeyGen}(1^\lambda)$: takes as input the security parameter λ and outputs a key pair $(\mathsf{sk}, \mathsf{pk})$. The public key pk includes a description of the message space $\mathcal{M}$.

$\mathsf{Enc}(\mathsf{pk}, m)$: takes as input pk and a message $m \in \mathcal{M}$, and it outputs a ciphertext C.

$\mathsf{Eval}(\mathsf{pk}, f, C_1, \ldots, C_t)$: the evaluation algorithm takes as input pk, a circuit $f : \mathcal{M}^t \to \mathcal{M}$ in a class $\mathcal{F}$ of "admissible" circuits, and t ciphertexts $C_1, \ldots, C_t$. It returns a ciphertext C.

$\mathsf{Dec}(\mathsf{sk}, C)$: given sk and a ciphertext C, the decryption algorithm outputs a message m.

A HE scheme is required to satisfy *correctness*, *compactness*, *security* and *circuit privacy*.

Definition 2 (Correctness). *A homomorphic encryption scheme $\mathsf{HE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ correctly evaluates a family of circuits $\mathcal{F}$ if for all honestly generated keys $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$, for all $f \in \mathcal{F}$ and for all messages $m_1, \ldots, m_t \in \mathcal{M}$ we have that if $C_i \leftarrow \mathsf{Enc}(\mathsf{pk}, m_i)$ $\forall i \in [t]$, then the following probability*

$$\Pr[\mathsf{Dec}(\mathsf{sk}, \mathsf{Eval}(\mathsf{pk}, f, (C_1, \ldots, C_t))) \neq f(m_1, \ldots, m_t)]$$

is negligible, where the probability is taken over all the algorithms' random choices.

Definition 3 (Compactness). *A HE scheme* $\mathsf{HE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ compactly evaluates *a family of circuits* $\mathcal{F}$ *if the running time of the decryption algorithm* Dec *is bounded by a fixed polynomial in* λ*.*

The security of a HE scheme is defined using the notion of semantic security.

Definition 4 (Semantic Security). *Let* $\mathsf{HE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *be a (homomorphic) encryption scheme, and* $\mathcal{A}$ *be a PPT adversary. Consider the following experiment:*

Experiment $\mathbf{Exp}^{\mathsf{SS}}_{\mathsf{HE},\mathcal{A}}(\lambda)$

$b \xleftarrow{\$} \{0,1\}$*;* $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$
$(m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pk})$
$c \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, m_b)$
$b' \leftarrow \mathcal{A}(c)$
If $b' = b$ *return 1. Else return 0.*

and define $\mathcal{A}$*'s advantage as* $\mathbf{Adv}^{\mathsf{SS}}_{\mathsf{HE},\mathcal{A}}(\lambda) = \Pr[\mathbf{Exp}^{\mathsf{SS}}_{\mathsf{HE},\mathcal{A}}(\lambda) = 1] - \frac{1}{2}$*. Then we say that* HE *is* semantically-secure *if for any PPT algorithm* $\mathcal{A}$ *it holds* $\mathbf{Adv}^{\mathsf{SS}}_{\mathsf{HE},\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda)$*.*

Finally, an homomorphic encryption scheme should satisfy circuit privacy. Roughly speaking, this property says that the ciphertexts output by Eval do not reveal any information about the messages encrypted in the input ciphertexts. This property is formally defined as follows:

Definition 5 (Circuit Privacy). *A homomorphic encryption scheme* HE *is* circuit private *for a family of circuits* $\mathcal{F}$ *if there exists a PPT simulator* Sim *and a negligible function* $\epsilon(\lambda)$ *such that the following holds. For any* $\lambda \in \mathbb{N}$*, any pair of keys* $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$*, any circuit* $f \in \mathcal{F}$*, any tuple of messages* $m_1, \ldots, m_t \in \mathcal{M}$ *and* $m = f(m_1, \ldots, m_t)$*, and ciphertexts* $C_1, \ldots, C_t$ *such that* $\forall i \in [t]$*:* $C_i \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, m_i)$*, then it holds*

$$\mathsf{SD}[\mathsf{Eval}(\mathsf{pk}, f, C_1, \ldots, C_t),\ \mathsf{Sim}(1^\lambda, \mathsf{pk}, m)] \leq \epsilon(\lambda)$$

3 Labeled HE

In this section we introduce the notion of *Labeled Homomorphic Encryption* (labHE, for short). This notion adapts the one of (symmetric-key) homomorphic encryption to the setting of labeled programs. This is based on the following key ideas. First, each piece of (encrypted) data that is outsourced is assigned a *unique* label which is used to identify the data. Second, whenever a client wants to ask the cloud to compute a function f on a portion of the outsourced (encrypted) data, the client specifies the inputs of f among the outsourced data. These inputs are identified by specifying their labels. The combination of f with these labels is called a *labeled program*. In short, labels allow clients to express queries on outsourced data.

In our homomorphic encryption notion, these ideas are introduced as follows. The encryption algorithm takes as input also a label; this is to say that the encryptor assigns a unique index to the encrypted data. Second, the decryption algorithm takes as additional input a labeled program; this is to express that the decryptor recovers the result of a known query (the labeled program) on the (unknown) outsourced data. In practice, the set of labels has concise representation (e.g. they can be names or even indexes in $[1, n]$).

Labeled Programs. Here we recall the notion of *labeled programs* [20], adapted to the case of arithmetic circuits as in [10]. The definition is taken almost verbatim from [10]. A *labeled program*

$\mathcal{P}$ is a tuple $(f, \tau_1, \ldots, \tau_n)$ such that $f : \mathcal{M}^n \to \mathcal{M}$ is a function on n variables (e.g., a circuit), and $\tau_i \in \{0,1\}^*$ is the label of the i-th variable input of f. Composition of labeled programs works as follows. Given the programs $\mathcal{P}_1, \ldots, \mathcal{P}_t$ and a function $g : \mathcal{M}^t \to \mathcal{M}$, the *composed program* $\mathcal{P}^*$ is obtained by evaluating g on the outputs of $\mathcal{P}_1, \ldots, \mathcal{P}_t$. Such a program is compactly denoted as $\mathcal{P}^* = g(\mathcal{P}_1, \ldots, \mathcal{P}_t)$. The (labeled) inputs of $\mathcal{P}^*$ are the all distinct labeled inputs of $\mathcal{P}_1, \ldots, \mathcal{P}_t$ (we assume that all inputs sharing the same label are grouped together in a single input of the new program). Let $f_{id} : \mathcal{M} \to \mathcal{M}$ be the canonical identity function and $\tau \in \{0,1\}^*$ be a label. Then $\mathcal{I}_\tau = (f_{id}, \tau)$ is the *identity program* for input label τ. With this notation in mind, notice that any labeled program $\mathcal{P} = (f, \tau_1, \ldots, \tau_n)$ can be expressed as the composition of n identity programs $\mathcal{P} = f(\mathcal{I}_{\tau_1}, \ldots, \mathcal{I}_{\tau_n})$.

Labeled Homomorphic Encryption A symmetric-key Labeled Homomorphic Encryption scheme labHE consists of the following algorithms.

$\mathsf{KeyGen}(1^\lambda)$. The key generation algorithm takes as input the security parameter λ. It outputs a secret key sk and a public evaluation key epk. We assume that epk implicitly contains a description of a message space $\mathcal{M}$, a label space $\mathcal{L}$, and a class $\mathcal{F}$ of "admissible" circuits.

$\mathsf{Enc}(\mathsf{sk}, \tau, m)$. The encryption algorithm takes as input the secret key sk, a label $\tau \in \mathcal{L}$ and a message $m \in \mathcal{M}$. It outputs a ciphertext C.

$\mathsf{Eval}(\mathsf{epk}, f, C_1, \ldots, C_t)$. On input epk, an arithmetic circuit $f : \mathcal{M}^t \to \mathcal{M}$ in the class $\mathcal{F}$ of "allowed" circuits, and t ciphertexts $C_1, \ldots, C_t$, the evaluation algorithm returns a ciphertext C.

$\mathsf{Dec}(\mathsf{sk}, \mathcal{P}, C)$. The decryption algorithm takes as input the secret key, a labeled program $\mathcal{P}$, and a ciphertext C, and it outputs a message $m \in \mathcal{M}$.

A labHE must satisfy *correctness, succinctness*, *semantic security*, and *context-hiding*.

Definition 6 (Correctness). *A Labeled Homomorphic Encryption scheme* $\mathsf{labHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *is said to* correctly evaluate *a family of circuits* $\mathcal{F}$ *if for all honestly generated keys* $(\mathsf{epk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$*, for all* $f \in \mathcal{F}$*, all labels* $\tau_1, \ldots, \tau_t \in \mathcal{L}$*, all messages* $m_1, \ldots, m_t \in \mathcal{M}$*, and any* $C_i \leftarrow \mathsf{Enc}(\mathsf{sk}, \tau_i, m_i)\ \forall i \in [t]$*,*

$$\Pr[\mathsf{Dec}(\mathsf{sk}, \mathcal{P}, \mathsf{Eval}(\mathsf{epk}, f, C_1, \ldots, C_t)) = f(m_1, \ldots, m_t)]$$

is negligibly close to 1, where $\mathcal{P} = (f, \tau_1, \ldots, \tau_t)$*. Probability is taken over the algorithms' random choices.*

Informally *succinctness* means that the size of ciphertexts output by Eval is some fixed polynomial in the security parameter, and does not depend on the size of the evaluated circuit. Formally, this is defined as follows.

Definition 7 (Succinctness). *A Labeled Homomorphic Encryption scheme* $\mathsf{labHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *is said to* succinctly evaluate *a family of circuits* $\mathcal{F}$ *if there is a fixed polynomial* $p(\cdot)$ *such that every honestly generated ciphertext (output of either* Enc *or* Eval*) has size (in bits)* $p(\lambda)$*.*

We note that our notion of succinctness is weaker than the notion of compactness of standard homomorphic encryption. Compactness dictates that the running time of the decryption algorithm is bounded by some fixed polynomial in λ. Succinctness is weaker in the sense that a compact scheme is also succinct whereas the converse might not be true (indeed our construction satisfies succinctness but not compactness).

The security of a labHE scheme is defined via a notion of semantic security that adapts to our setting the standard notion put forward by Goldwasser and Micali [23].

Definition 8 (Semantic Security for labHE). *Let* $\mathsf{labHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *be a Labeled Homomorphic Encryption scheme and* $\mathcal{A}$ *be a PPT adversary. Consider the following experiment where* $\mathcal{A}$ *is given access to an oracle* $\mathsf{Enc}(\mathsf{sk}, \cdot, \cdot)$ *that on input a pair* (τ, m) *outputs* $\mathsf{Enc}(\mathsf{sk}, \tau, m)$*:*

Experiment $\mathbf{Exp}^{\mathsf{SS}}_{\mathsf{labHE},\mathcal{A}}(\lambda)$

$b \xleftarrow{\$} \{0,1\}$; $(\mathsf{epk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$
$(m_0, \tau_0^*, m_1, \tau_1^*) \leftarrow \mathcal{A}^{\mathsf{Enc}(\mathsf{sk},\cdot,\cdot)}(\mathsf{epk})$
$c \xleftarrow{\$} \mathsf{Enc}(\mathsf{sk}, \tau_b^*, m_b)$
$b' \leftarrow \mathcal{A}^{\mathsf{Enc}(\mathsf{sk},\cdot,\cdot)}(c)$
If $b' = b$ *return 1. Else return 0.*

We say that $\mathcal{A}$ *is a* legitimate *adversary if it queries the encryption oracle on distinct labels (i.e., each label* τ *is never queried more than once), and never on the two challenge labels* τ_0^*, τ_1^**. We define* $\mathcal{A}$*'s advantage as* $\mathbf{Adv}^{\mathsf{SS}}_{\mathsf{labHE},\mathcal{A}}(\lambda) = \Pr[\mathbf{Exp}^{\mathsf{SS}}_{\mathsf{labHE},\mathcal{A}}(\lambda) = 1] - \frac{1}{2}$*. Then we say that* labHE *provides* semantic-security *if for any PPT legitimate algorithm* $\mathcal{A}$ *it holds* $\mathbf{Adv}^{\mathsf{SS}}_{\mathsf{labHE},\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda)$*.*

Finally we define another security property of Labeled Homomorphic Encryption that we call *context-hiding*. In a nutshell, context-hiding says that a user running $m = \mathsf{Dec}(\mathsf{sk}, \mathcal{P}, C)$ learns nothing about the input m', except that $m = f(m')$, where f is the function in $\mathcal{P}$.

Definition 9 (Context Hiding). *We say that a Labeled Homomorphic Encryption scheme* labHE *satisfies* context-hiding *for a family of circuits* $\mathcal{F}$ *if there exists a PPT simulator* Sim *and a negligible function* $\epsilon(\lambda)$ *such that the following holds. For any* $\lambda \in \mathbb{N}$*, any pair of keys* $(\mathsf{epk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$*, any circuit* $f \in \mathcal{F}$ *with* t *inputs, any tuple of messages* $m_1, \ldots, m_t \in \mathcal{M}$*, labels* $\tau_1, \ldots, \tau_t \in \mathcal{L}$*, and corresponding ciphertexts* $C_i \leftarrow \mathsf{Enc}(\mathsf{sk}, \tau_i, m_i)$ $\forall i = 1, \ldots, t$*, we have that*

$$\mathsf{SD}[\mathsf{Eval}(\mathsf{epk}, f, C_1, \ldots, C_t),\ \mathsf{Sim}(1^\lambda, \mathsf{sk}, \mathcal{P}, m)] \leq \epsilon(\lambda)$$

where $\mathcal{P} = (f, \tau_1, \ldots, \tau_t)$ *and* $m = f(m_1, \ldots, m_t)$*.*

Labeled Homomorphic Encryption with Preprocessing. Here we define a special case of Labeled Homomorphic Encryption where some of the algorithms allow for a preprocessing step that enables to speed up online computations.

We say that a scheme labHE has *offline/online encryption* if it admits two algorithms Offline-Enc and Online-Enc working as follows:

Offline-Enc(sk, τ). This algorithm takes as input a label and the secret key and produces an offline ciphertext C_{off} for the label τ.

Online-Enc(C_{off}, m). On input a message m and an offline ciphertext for the label τ the algorithm produces a ciphertext C.

The two algorithms must be correct in the sense that $\mathsf{Enc}(\mathsf{sk}, \tau, m)$ equals the outcome of Online-Enc(Offline-Enc$(\mathsf{sk}, \tau), m)$. Informally, the first algorithm is the computationally more costly procedure that can be run independently of the actual message one wishes to encrypt. Online-Enc, on the other hand, is more efficient but can be executed only when m becomes available.

Second, we say that a scheme labHE has *offline/online decryption* if it admits two algorithms Offline-Dec and Online-Dec working as follows:

$\mathsf{Offline\text{-}Dec}(\mathsf{sk}, \mathcal{P})$. This algorithm takes as input the secret key and the labeled program and produces an offline secret key $\mathsf{sk}_{\mathsf{off}}$ for $\mathcal{P}$. Notice that $\mathsf{sk}_{\mathsf{off}}$ does not depend on any specific ciphertext.

$\mathsf{Online\text{-}Dec}(\mathsf{sk}_{\mathsf{off}}, C)$. On input $\mathsf{sk}_{\mathsf{off}}$ and C it outputs a message a message m.

Again, the two algorithms must be correct in the sense that $\mathsf{Dec}(\mathsf{sk}, \mathcal{P}, C)$ equals the outcome of $\mathsf{Online\text{-}Dec}(\mathsf{Offline\text{-}Dec}(\mathsf{sk}, \mathcal{P}), C)$. Informally, offline/online decryption allows to split the decryption procedure into two parts: the offline one which is computationally more expensive and may depend on the complexity of the program $\mathcal{P}$; the online part that is much faster and whose running time is a fixed polynomial in the security parameter.

4 A Construction of Labeled HE for Quadratic Polynomials

In this section we present a construction of Labeled Homomorphic Encryption that supports the evaluation of degree-two polynomials. Our construction builds upon the technique of [11] for boosting linearly homomorphic encryption schemes to evaluate degree-two polynomials on ciphertexts. Interestingly, however, while the construction from [11] achieves succinctness only for the subclass of degree-two polynomials where the number of degree-two monomials is bounded by a constant, our realization achieves succinctness for *all* degree-two polynomials. Similarly to [11], our realization builds upon any (linearly) homomorphic encryption scheme that is *public space* (e.g., [34]). This property requires that the message space $\mathcal{M}$ is a (publicly known) commutative ring where it is possible to sample random elements efficiently (see [11] for a more rigorous definition).

Let $\hat{\mathsf{HE}} = (\hat{\mathsf{KeyGen}}, \hat{\mathsf{Enc}}, \hat{\mathsf{Eval}}, \hat{\mathsf{Dec}})$ be a public-space linearly-homomorphic encryption scheme (see [21] for the details). Following [11] we denote with $\hat{\mathcal{C}}$ the ciphertext space of $\hat{\mathsf{HE}}$, we use Greek letters to denote elements of $\hat{\mathcal{C}}$ and Roman letters for elements of $\mathcal{M}$. Without loss of generality we assume that $\hat{\mathsf{Eval}}$ consists of two procedures: one to perform (homomorphic) additions and another to perform (homomorphic) multiplications by constants. We denote these operations with $\boxplus$ and $\cdot$, respectively and (abusing notation) we denote addition and multiplication in $\mathcal{M}$ as $+$ and $\cdot$.

We propose a Labeled Homomorphic Encryption scheme $\mathsf{labHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ capable of evaluating multivariate polynomials of degree 2 over $\mathcal{M}$, with respect to some (finite) set of labels $\mathcal{L} \subset \{0,1\}^*$. We use a pseudorandom function $F : \{0,1\}^k \times \{0,1\}^* \to \mathcal{M}$, with key space $\{0,1\}^k$, for some $k = \mathsf{poly}(\lambda)$.

$\mathsf{KeyGen}(1^\lambda)$: On input a security parameter $\lambda \in \mathbb{N}$, run $\hat{\mathsf{KeyGen}}(1^\lambda)$ to get $(\mathsf{pk}, \mathsf{sk}')$. Next, choose a random seed $K \in \{0,1\}^k$ for the PRF, and set $\mathcal{L} = \{0,1\}^*$. Output $\mathsf{sk} = (\mathsf{sk}', K)$ and $\mathsf{epk} = (\mathsf{pk}, \mathcal{L})$. The above assumes that pk already describes both $\hat{\mathsf{HE}}$'s message space $\mathcal{M}$ and its ciphertext space $\hat{\mathcal{C}}$. The message space of labHE will be $\mathcal{M}$.

$\mathsf{Enc}(\mathsf{sk}, \tau, m)$: We describe Enc directly in terms of its two components $\mathsf{Offline\text{-}Enc}$ and $\mathsf{Online\text{-}Enc}$.

$\mathsf{Offline\text{-}Enc}(\mathsf{sk}, \tau)$: Given a label τ, compute $b \leftarrow F(K, \tau)$ and outputs $C_{\mathsf{off}} = (b, \hat{\mathsf{Enc}}(\mathsf{pk}, b))$.

$\mathsf{Online\text{-}Enc}(C_{\mathsf{off}})$ Parse C_{off} as (b, β) and output $C = (a, \beta)$, where $a \leftarrow m - b$ (in $\mathcal{M}$). Notice that the cost of online encryption is that of an addition in $\mathcal{M}$.

$\mathsf{Eval}(\mathsf{epk}, f, C_1, \ldots, C_t)$: Eval is composed of 3 different different procedures: $\mathsf{Mult}, \mathsf{Add}, \mathsf{cMult}$. We describe each such procedure separately. Informally, Mult allows to perform (homomorphic) multiplications, Add deals with homomorphic additions and cMult takes care of (homomorphic) multiplications by known constants.

Mult : On input two ciphertexts $C'_1, C'_2 \in \mathcal{M} \times \hat{\mathcal{C}}$ where, for $i = 1, 2$, $C_i = (a_i, \beta_i)$, the algorithm computes a "multiplication" ciphertext $C = \alpha \in \hat{\mathcal{C}}$ as:

$$\alpha = \hat{\mathsf{Enc}}(\mathsf{pk}, a_1 \cdot a_2) \boxplus a_1 \cdot \beta_2 \boxplus a_2 \cdot \beta_1$$

Correctness follow from the fact that, if $a_i = (m_i - b_i)$ and $\beta_i \in \hat{\mathsf{Enc}}(\mathsf{pk}, b_i)$ for some $b_i \in \mathcal{M}$, then

$$\alpha \in \hat{\mathsf{Enc}}\,(\mathsf{pk}, (m_1 m_2 - b_1 m_2 - b_2 m_1 + b_1 b_2) + (b_2 m_1 - b_1 b_2) + (b_1 m_2 - b_1 b_2)) = \hat{\mathsf{Enc}}(\mathsf{pk}, m_1 m_2 - b_1 b_2)$$

Add : We distinguish two cases depending on the format of the two input ciphertexts C_1, C_2. If $C_1, C_2 \in \mathcal{M} \times \hat{\mathcal{C}}$ where, for $i = 1, 2$, $C_i = (a_i, \beta_i)$, then the algorithm produces a new ciphertext $C = (a, \beta) \in \mathcal{M} \times \hat{\mathcal{C}}$ computed as

$$a = a_1 + a_2, \quad \beta = \beta_1 \boxplus \beta_2$$

For correctness in this case note that if $a_i = (m_i - b_i)$ and $\beta_i \in \hat{\mathsf{Enc}}(\mathsf{pk}, b_i)$ for some $b_i \in \mathcal{M}$, then $a = (m_1 + m_2) - (b_1 + b_2)$ and $\beta \in \hat{\mathsf{Enc}}(\mathsf{pk}, b_1 + b_2)$.
If, on the other hand, the received ciphertexts are $C_1, C_2 \in \hat{\mathcal{C}}$ where, for $i = 1, 2$, $C_i = \alpha_i$, the new ciphertext $C = \alpha \in \hat{\mathcal{C}}$ is computed as $\alpha = \alpha_1 \boxplus \alpha_2$.

cMult : As before, on input a constant $c \in \mathcal{M}$ and a ciphertext C, we distinguish two cases depending on the format of C.
If $C = (a, \beta) \in \mathcal{M} \times \hat{\mathcal{C}}$, this algorithm returns a ciphertext $C' = (a \cdot c, c \cdot \beta) \in \mathcal{M} \times \hat{\mathcal{C}}$.
If, on the other hand, $C = \alpha \in \hat{\mathcal{C}}$, this algorithm returns a ciphertext $C' = c \cdot \alpha \in \hat{\mathcal{C}}$.
The correctness of the above operations is straightforward.

$\mathsf{Dec}(\mathsf{sk}, \mathcal{P}, C)$: As for the case of the encryption procedure, we describe the algorithm in terms of its two components $\mathsf{Offline\text{-}Dec}$ and $\mathsf{Online\text{-}Dec}$.

$\mathsf{Offline\text{-}Dec}(\mathsf{sk}, \mathcal{P})$ Given sk and the labeled program $\mathcal{P}$, parse $\mathcal{P}$ as $(f, \tau_1, \ldots, \tau_t)$. For $i = 1, \ldots, t$, the algorithm computes $b_i \leftarrow F(K, \tau_i)$, $b = f(b_1, \ldots, b_t)$ and outputs $\mathsf{sk}_{\mathcal{P}} = (\mathsf{sk}, b)$.

$\mathsf{Online\text{-}Dec}(\mathsf{sk}_{\mathcal{P}}, C)$ Parse $\mathsf{sk}_{\mathcal{P}}$ as (sk, b), we distinguish two cases depending on whether $C \in \mathcal{M} \times \hat{\mathcal{C}}$ or not.
If $C = (a, \beta) \in \mathcal{M} \times \hat{\mathcal{C}}$ there are two decryption methods: (i) output $m = a + b$; (ii) output $m = a + \hat{\mathsf{Dec}}(\mathsf{sk}, \beta)$.
If $C \in \hat{\mathcal{C}}$ set $\hat{m} = \hat{\mathsf{Dec}}(\mathsf{sk}, C)$ and output $m = \hat{m} + b$.
Notice that the cost of online decryption solely depends on the cost of $\hat{\mathsf{Dec}}$ and it is totally independent of $\mathcal{P}$. Moreover the decryption method (ii) does not require the offline phase.

Correctness. The overall correctness of labHE follows from the following facts:

- For every (τ, m), we have $\mathsf{Enc}(\mathsf{sk}, \tau, m) \to C = (a, \beta)$ where $a = m - f_{id}(F(K, \tau))$. Therefore, by construction and correctness of $\hat{\mathsf{HE}}$, we have $m \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathcal{I}_\tau, C)$.
- For $i = 1, 2$, consider any two labeled programs $\mathcal{P}_i = (f_i, \tau_1^{(i)}, \ldots, \tau_{t_i}^{(i)})$ and ciphertexts C_i such that $m_i \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathcal{P}_i, C_i)$. Then for any gate g, we have that $\mathsf{Eval}(\mathsf{epk}, g, C_1, C_2)$ outputs a ciphertext C such that $\mathsf{Dec}(\mathsf{sk}, \mathcal{P}^*, C) \to g(m_1, m_2)$, where $\mathcal{P}^* = g(\mathcal{P}_1, \mathcal{P}_2)$. To see this observe that: (1) by construction a ciphertext C_i is either $(m_i - b_i, \hat{\mathsf{Enc}}(b_i))$ or $\hat{\mathsf{Enc}}(m_i - b_i)$ where $b_i = f_i(F(K, \tau_1^{(i)}), \ldots, F(K, \tau_{t_i}^{(i)}))$; (2) after every homomorphic evaluation of gate g we obtain a ciphertext C that is either $(g(m_1, m_2) - g(b_1, b_2), \hat{\mathsf{Enc}}(g(b_1, b_2)))$ or $\hat{\mathsf{Enc}}(g(m_1, m_2) - g(b_1, b_2))$. By construction of Dec, clearly we have $\mathsf{Dec}(\mathsf{sk}, \mathcal{P}^*, C) \to g(m_1, m_2)$.

To see the the above facts, recall from Section 3 the various properties of labeled programs, e.g., that f_{id} is the identity function, $\mathcal{I}_\tau = (f_{id}, \tau)$ is the identity program with label τ, and $\mathcal{P}* = g(\mathcal{P}_1, \mathcal{P}_2)$ is the composition of labeled programs.

Security. The following two theorems prove that our labHE scheme satisfies semantic security and context hiding respectively.

Theorem 1. *If* $\hat{\mathsf{HE}}$ *is semantically-secure and* F *is pseudorandom then* labHE *is semantically secure.*

The proof is obtained via a simple hybrid argument. First, notice that if one modifies $\mathbf{Exp}^{\mathsf{SS}}_{\mathsf{labHE},\mathcal{A}}(\lambda)$ so that the b's corresponding to τ_0 and τ_1 are taken at random (rather than using F), then the resulting experiment is computationally indistinguishable from the original one, under the assumption that F is PRF. Afterwards, notice that

$$\begin{aligned}(m_0 - b_0, \hat{\mathsf{Enc}}(\mathsf{pk}, b)) &\approx (m_0 - b_0, \hat{\mathsf{Enc}}(\mathsf{pk}, 0)) \\ \equiv (m_1 - b_1, \hat{\mathsf{Enc}}(\mathsf{pk}, 0)) &\approx (m_1 - b_1, \hat{\mathsf{Enc}}(\mathsf{pk}, b_1))\end{aligned}$$

where $\approx$ denotes computational indistinguishability by the semantic security of $\hat{\mathsf{HE}}$ and $\equiv$ means that the distributions are identical.

Theorem 2. *If* $\hat{\mathsf{HE}}$ *is circuit-private, then* labHE *is context-hiding.*

Proof. We prove the theorem by showing the following simulator. Let $\hat{\mathsf{Sim}}$ be the simulator for the circuit privacy of $\hat{\mathsf{HE}}$. If f is a degree-1 polynomial the simulator $\mathsf{Sim}(1^\lambda, \mathsf{sk}, (f, \tau_1, \ldots, \tau_t), m)$ computes $b = f(F(K, \tau_1), \ldots, F(K, \tau_t))$ and outputs $C = (m - b, \hat{\mathsf{Sim}}(1^\lambda, \mathsf{pk}, b))$. If f is of degree 2, the simulator does the same except that it computes $C = \hat{\mathsf{Sim}}(1^\lambda, \mathsf{pk}, m - b)$. It is straightforward to see that by the circuit privacy of $\hat{\mathsf{HE}}$ C is distributed identically to the ciphertext produced by Eval.

5 Multi-User Labeled HE

In this section we introduce a multi-user variant of Labeled Homomorphic Encryption. The main idea is that encryptors do not share a global common secret key. Rather, each user i employs his own secret key usk_i to encrypt, yet it is possible to homomorphically compute over data encrypted by different users. Decryption then requires knowledge of the master secret along with the public keys of all the users whose ciphertexts were involved in the computation.

A Multi-User Labeled Homomorphic Encryption scheme consists of a tuple of algorithms $\mathsf{mu\text{-}labHE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ working as follows.

$\mathsf{Setup}(1^\lambda)$. The setup algorithm takes as input the security parameter λ, and outputs a master secret key msk and a master public key mpk. We assume that mpk implicitly contains a description of a message space $\mathcal{M}$, a label space $\mathcal{L}$, and a class $\mathcal{F}$ of "admissible" circuits.

$\mathsf{KeyGen}(\mathsf{mpk})$. The key generation algorithm takes as input the master public key mpk and outputs a user secret key usk and a user public key upk.

$\mathsf{Enc}(\mathsf{mpk}, \mathsf{usk}, \tau, m)$. The encryption algorithm takes as input the master public key mpk, a user secret key usk, a label $\tau \in \mathcal{L}$ and a message $m \in \mathcal{M}$. It outputs a ciphertext C.

$\mathsf{Eval}(\mathsf{mpk}, f, C_1, \ldots, C_t)$ On input mpk, an arithmetic circuit $f : \mathcal{M}^t \to \mathcal{M}$ in the class $\mathcal{F}$ of "allowed" circuits, and t ciphertexts $C_1, \ldots, C_t$, the evaluation algorithm returns a ciphertext C.

$\mathsf{Dec}(\mathsf{sk}, \mathbf{upk}, \mathcal{P}, C)$ The decryption algorithm takes as input the secret key, a vector of user secret keys $\mathbf{upk} = (\mathsf{upk}_1, \ldots, \mathsf{upk}_\ell)$, a labeled program $\mathcal{P}$, and a ciphertext C, and it outputs a message $m \in \mathcal{M}$.

A Multi-User Labeled Homomorphic Encryption scheme is required to satisfy *correctness, succinctness*, *semantic security*, and *context-hiding* as defined below.

Definition 10 (Correctness). *A Multi-User Labeled Homomorphic Encryption scheme* $\mathsf{mu\text{-}labHE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ correctly evaluates *a family of circuits* $\mathcal{F}$ *if for all honestly generated keys* $(\mathsf{mpk}, \mathsf{msk}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$*, all user keys* $(\mathsf{upk}_1, \mathsf{usk}_1), \ldots, (\mathsf{upk}_\ell, \mathsf{usk}_\ell) \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{mpk})$*, for all* $f \in \mathcal{F}$*, all labels* $\tau_1, \ldots, \tau_t \in \mathcal{L}$*, messages* $m_1, \ldots, m_t \in \mathcal{M}$*, and any* $C_i \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{usk}_{j_i}, \tau_i, m_i)$ $\forall i \in [t], j_i \in [\ell]$*:*

$$\Pr[\mathsf{Dec}(\mathsf{sk}, \mathbf{upk}, \mathcal{P}, \mathsf{Eval}(\mathsf{pk}, f, C_1, \ldots, C_t)) = f(m_1, \ldots, m_t)]$$

is negligibly close to 1, where $\mathcal{P} = (f, \tau_1, \ldots, \tau_t)$*. Probability is taken over the algorithms' random choices.*

The notion of succinctness for multi-user Labeled Homomorphic Encryption is identical to that given in Definition 7. Security of Multi-User Labeled Homomorphic Encryption is defined similarly to that of labHE.

Definition 11 (Semantic Security for mu-labHE). *Let* $\mathsf{mu\text{-}labHE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *be a Multi-User Labeled Homomorphic Encryption scheme and* $\mathcal{A}$ *be a PPT adversary. Consider the following experiment where* $\mathcal{A}$ *is given access to an oracle* $\mathsf{Enc}(\mathsf{mpk}, \mathsf{usk}, \cdot, \cdot)$ *that on input a pair* (τ, m) *outputs* $\mathsf{Enc}(\mathsf{mpk}, \mathsf{usk}, \tau, m)$*:*

Experiment $\mathbf{Exp}^{\mathsf{SS}}_{\mathsf{mu\text{-}labHE},\mathcal{A}}(\lambda)$
$b \xleftarrow{\$} \{0,1\}$*;* $(\mathsf{mpk}, \mathsf{msk}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$*;*
$(\mathsf{upk}, \mathsf{usk}) \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{mpk})$
$(m_0, \tau_0^*, m_1, \tau_1^*) \leftarrow \mathcal{A}^{\mathsf{Enc}(\mathsf{mpk},\mathsf{usk},\cdot,\cdot)}(\mathsf{mpk}, \mathsf{upk})$
$C \xleftarrow{\$} \mathsf{Enc}(\mathsf{mpk}, \mathsf{usk}, \tau_b^*, m_b)$
$b' \leftarrow \mathcal{A}^{\mathsf{Enc}(\mathsf{mpk},\mathsf{usk},\cdot,\cdot)}(C)$
If $b' = b$ *return 1. Else return 0.*

We say that $\mathcal{A}$ *is a* legitimate *adversary if it queries the encryption oracle on distinct labels (i.e., each label* τ *is never queried more than once), and never on the two challenge labels* τ_0^*, τ_1^**. We define* $\mathcal{A}$*'s advantage as* $\mathbf{Adv}^{\mathsf{SS}}_{\mathsf{mu\text{-}labHE},\mathcal{A}}(\lambda) = \Pr[\mathbf{Exp}^{\mathsf{SS}}_{\mathsf{mu\text{-}labHE},\mathcal{A}}(\lambda) = 1] - \frac{1}{2}$*. Then we say that* mu-labHE *has* semantic-security *if for any PPT legitimate algorithm* $\mathcal{A}$ *it holds* $\mathbf{Adv}^{\mathsf{SS}}_{\mathsf{mu\text{-}labHE},\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda)$*.*

Finally we adapt the notion of *context-hiding* of Labeled Homomorphic Encryption to the multi-user case. The intuitive meaning of the notion is the same, namely the decryptor does not learn more information beyond what is in the input/output of the decryption algorithm.

Definition 12 (Context Hiding). *A Multi-User Labeled Homomorphic Encryption scheme* mu-labHE *satisfies* context-hiding *for a family of circuits* $\mathcal{F}$ *if there exists a PPT simulator* Sim *and a negligible function* $\epsilon(\lambda)$ *such that the following holds. For any* $\lambda \in \mathbb{N}$*, any pair of master keys* $(\mathsf{mpk}, \mathsf{msk}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$*, any* ℓ *user keys* $(\mathsf{upk}_1, \mathsf{usk}_1), \ldots, (\mathsf{upk}_\ell, \mathsf{usk}_\ell) \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{mpk})$*, any circuit*

$f \in \mathcal{F}$ with t inputs, any tuple of messages $m_1, \ldots, m_t \in \mathcal{M}$, labels $\tau_1, \ldots, \tau_t \in \mathcal{L}$, and ciphertexts $C_i \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{usk}_{j_i}, \tau_i, m_i)$ $\forall i = 1, \ldots, t$ and $j_i \in [\ell]$:

$$\mathsf{SD}[\mathsf{Eval}(\mathsf{epk}, f, C_1, \ldots, C_t),\ \mathsf{Sim}(1^\lambda, \mathsf{msk}, \mathbf{upk}, \mathcal{P}, m)] \leq \epsilon(\lambda)$$

where $\mathcal{P} = (f, \tau_1, \ldots, \tau_t)$ and $m = f(m_1, \ldots, m_t)$.

5.1 mu-labHE for Quadratic Polynomials

Here we show that the scheme of Section 4 can be easily modified to become multi-user.

$\mathsf{Setup}(1^\lambda)$: On input a security parameter $1^\lambda \in \mathbb{N}$, it runs $\hat{\mathsf{KeyGen}}(1^\lambda)$ to get $(\mathsf{pk}, \mathsf{sk})$. Set $\mathcal{L} = \{0,1\}^*$. Output $(\mathsf{msk} = \mathsf{sk}, \mathsf{mpk} = \mathsf{pk})$. The above assumes that pk already describes both $\hat{\mathsf{HE}}$'s message space $\mathcal{M}$ and its ciphertext space $\hat{\mathcal{C}}$. The message space for mu-labHE will be $\mathcal{M}$.

$\mathsf{KeyGen}(\mathsf{mpk})$: On input the master public key, the key generation algorithm samples a random seed $K \xleftarrow{\$} \{0,1\}^k$, computes $C_K \xleftarrow{\$} \hat{\mathsf{Enc}}(\mathsf{mpk}, K)$, and outputs $(\mathsf{upk}, \mathsf{usk})$, where $\mathsf{upk} = C_K$ and $\mathsf{usk} = K$.

$\mathsf{Enc}(\mathsf{mpk}, \mathsf{usk}, \tau, m)$: Enc has two components Offline-Enc and Online-Enc.

$\mathsf{Offline\text{-}Enc}(\mathsf{mpk}, \mathsf{usk}, \tau)$: Given a label τ, compute $b \leftarrow F(\mathsf{usk}, \tau)$ and outputs $C_{\mathsf{off}} = (b, \hat{\mathsf{Enc}}(\mathsf{mpk}, b))$.

$\mathsf{Online\text{-}Enc}(C_{\mathsf{off}})$ Parse C_{off} as (b, β), output $C = (a, \beta)$, where $a \leftarrow m - b \in \mathcal{M}$.

$\mathsf{Eval}(\mathsf{pk}, f, C_1, \ldots, C_t)$: This algorithm is identical to that described in Section 4.

$\mathsf{Dec}(\mathsf{msk}, \mathbf{upk}, \mathcal{P}, C)$: As for the case of the encryption procedure, we describe the algorithm in terms of its two components Offline-Dec and Online-Dec.

$\mathsf{Offline\text{-}Dec}(\mathsf{msk}, \mathbf{upk}, \mathcal{P})$ Let $\mathbf{upk} = (\mathsf{upk}_1, \ldots, \mathsf{upk}_\ell)$. For all $i \in [\ell]$, decrypt $\mathsf{usk}_i \leftarrow \hat{\mathsf{Dec}}(\mathsf{msk}, \mathsf{upk}_i)$. Next, parse the labeled program $\mathcal{P}$, parse $\mathcal{P}$ as $(f, \tau_1, \ldots, \tau_t)$. For $i = 1, \ldots, t$, the algorithm computes $b_i \leftarrow F(\mathsf{usk}_{j_i}, \tau_i)$, $b = f(b_1, \ldots, b_t)$ and outputs $\mathsf{sk}_{\mathcal{P}} = (\mathsf{sk}, b)$. Notice that for every $i \in [t]$ the index $j_i \in [\ell]$ is assumed to be part of the description of the labeled program. Namely, one knows to which input wire each user contributed.

$\mathsf{Online\text{-}Dec}(\mathsf{sk}_{\mathcal{P}}, C)$ Identical to the corresponding algorithm of Section 4.

The succinctness of mu-labHE can be checked by inspection. Correctness and semantic security of mu-labHE are the same as for labHE. Context hiding is also very similar, with the only difference that the simulator has to first obtain usk_i by decrypting upk_i.

6 Statistics using labHE

In this section we show that by using our constructions of (multi-user) Labeled Homomorphic Encryption for quadratic polynomials, it is possible to compute relevant statistical functions over encrypted data. In the next Section we will then describe two application scenarios where the specific features of our protocol act as enablers for real-world applications. Intuitively, the restriction of computing only quadratic polynomials can be described as follows: suppose a value x and a value y are secret and are encrypted using our scheme. Then, one can compute any polynomial of the form $a_1x^2 + a_2y^2 + a_3xy + a_4x + a_5y + a_6$. More generally, given an arbitrary number of encrypted values, possibly coming from many users, one can compute any function that can be expressed as a linear function of those values and pairwise products between those values. We will see a few interesting examples of this next.

Consider a *dataset* as a matrix $X = \{x_{i,j}\}$, for $i = 1, \ldots, n$ and $j = 1, \ldots, d$. Number d represents the dimension (i.e., the number of variables/columns) while n is the number of dataset members (or rows).

Mean and Covariance First, we show how to compute the *mean* and *covariance* over a multi-dimensional dataset X. It is not hard to see how to extend these ideas to the computation of any other function that can be represented with a degree-2 polynomial. Such functions include, e.g., the root mean square (RMS), and the Pearson's and uncentered correlation coefficient.

The mean of the j-th column is the value $\mu_j = \frac{1}{n}\sum_{i=1}^{n} x_{i,j}$. Since our labHE does not support division, we will use it to compute homomorphically the value $\hat{\mu}_j = \sum_{i=1}^{n} x_{i,j}$ and let the receiver do the division after decryption. This is a natural thing to do in scenarios where the computation conducted over the data is known to the decryptor, which is something that labelled homomorphic encryption implicitly assumes.

For a dataset X, its covariance matrix $C = \{c_{j,k}\}$ for $j, k = 1, \ldots, d$ is defined as

$$c_{j,k} = \frac{1}{n}\sum_{i=1}^{n} x_{i,j} \cdot x_{i,k} - \frac{1}{n^2}\left(\sum_{i=1}^{n} x_{i,j}\right)\left(\sum_{i=1}^{n} x_{i,k}\right)$$

Again we will use the scheme to compute homomorphically the integers

$$\hat{c}_{j,k} = n^2 \cdot c_{j,k} = n\sum_{i=1}^{n} x_{i,j} \cdot x_{i,k} - \left(\sum_{i=1}^{n} x_{i,j}\right)\left(\sum_{i=1}^{n} x_{i,k}\right)$$

and let the receiver obtain $c_{j,k}$ by doing a division by n^2 after decryption.

Weighted Sum Given a dataset $X = \{x_{i,j}\}$ and a vector of weights $\boldsymbol{y} = \{y_i\}_{i=1}^{n}$, the *weighted sum* of the j-th column of X is the value $\omega_j = \sum_{i=1}^{n} x_{i,j} \cdot y_i$.

There are two situations to consider. If the weights are *not* secret, then the weighted sum can be expressed as a degree-1 polynomial over the encrypted column X. If, on the other hand, the vector of weights is itself secret, then the weighted sum becomes a degree two polynomial (an inner-product) between two vectors of encrypted values. We will see in the next section how this can be useful for genenetic association tests.

Euclidean Distance Given a matrix $X = \{x_{i,j}\}$ the (square of) Euclidean distance between the j-th column of X and a vector $\boldsymbol{y} = \{y_i\}_{i=1}^{n}$ is the value $\delta_j = \sum_{i=1}^{n}(x_{i,j} - y_i)^2$. This is an example of a function that requires a quadratic computation if either part of the data set is encrypted.

7 Applications and evaluation

We have implemented our (multi-user) labHE realization in C starting from the GNU Multiprecision Library[4] (GMP) and the Kekkac Code Package[5] (KCP). We used GMP to implement the linearly homomorphic encryption scheme by Joye and Libert [28] (JL13) and relied on Sponge-based pseudorandom function included in the KCP. The JL13 cryptosystem has message space $\mathbb{Z}_{2^k}$ and works over $\mathbb{Z}_N^*$, where $N = pq$ is the product of two quasi-safe primes $p = 2^k p' + 1$ and $q = 2^k q' + 1$. For security [28] k needs to be at most $1/4 \log N - \lambda$, where λ is the security parameter. Note that taking message space $\mathbb{Z}_{2^k}$ allows to perform computations over the integers with k-bits precision, and also to encode real values by using fixed point representations with suitable scaling as described, e.g., in [13]. Although our implementation is flexible, we fixed the security level at that of 2048 RSA moduli, conjectured to correspond to roughly 100-112 bits of security.[6] The message space

[4] `https://gmplib.org/`
[5] `https://github.com/gvanas/KeccakCodePackage`
[6] `https://www.keylength.com/`

was chosen to be that of bit strings of 128 bits. All our implementations are single-threaded. The implementation is available upon request.

Our benchmarking results were collected in a standard MacBook Pro machine with a 2.7 GHz Intel Core i5 and 16 GB or RAM. For every chosen set of parameters, we repeated the experiment 10 times, and took the median of the timings. In all cases we observed a coefficient of variation below 10%.

Micro-benchmarks We first discuss the communication/storage costs of our solution. Every ciphertext of our scheme, instantiated with the above parameters can be encoded into 272 bytes. For instance, if we consider a dataset with $n = 2^{20}$ rows and $d = 2$ columns, it means that a server has to store about 560 MBytes.

We now turn to the timings of basic operations such as key generation, encryption and decryption of level-1 ciphertexts (i.e., outputs of degree-1 functions, such as the Mean). Collected timings are summarized in Table 1. Notably, while key generation is relatively relevant (it is executed only once), the speed in the encryption procedure (that is executed for every dataset item) is way more relevant for scalability. For a large data size such as the one above, encryption can be done in 12 minutes in a modest machine. As we detail in Appendix A, the encryption time of state-of-the-art somewhat homomorphic encryption (for the same class of functions) is 85× slower, meaning that encrypting a dataset of the above size would take about 17 hours.

Table 1. Timings of micro-benchmarks.

Operation	Time (ms)
KeyGen	155.11
Enc	0.35
Dec	3.42

7.1 Application #1: Privacy Preserving Statistics on Outsourced Data

We now consider a real-world secure statistics scenario, where a potentially large data set is stored in an untrusted Cloud and a Client can ask for the Cloud to perform statistical computations over encrypted data.

Security model There are two variations for this scenario, with different trust models associated with them.

The first scenario is the classical setting of computation outsourcing from a computationally limited Client to a powerful worker. Namely, the Client acts as both the Data Provider and the Receiver. In this case, the trust relation involves the Client and the Cloud only, as one can assume that data is encrypted by the Client itself and the Cloud is used as a computational resource, which is assumed to be honest-but-curious.

The second scenario involves three agents, and the trust relations are more complex. A Data Provider is willing to allow the Client to perform statistical computations on the data and learn the results, but not to access the raw data itself. To this end, it encrypts the data and outsources it to the Cloud, which is trusted not to provide this encrypted data to the Client (as otherwise the raw data would be revealed). The Client trusts the Data Provider to encrypt the correct data, and the Cloud to follow the protocol.

The advantage of our protocol in this case, is that it permits achieving unprecedented levels of efficiency for the statistical functions we have shown to fall within the class of computations that can be encoded as degree two polynomials. We emphasize that these applications are excellent examples of the advantages of our scheme, since they do *not* require hiding the details of the computation that is being carried out from the Client, although they may require hiding the originally encrypted input values. It is clear that (non-labelled somewhat) homomorphic encryption could be used for the same purpose, and it could even hide the computation being carried out, but our results show that this would be a costly overkill (cf. Appendix A).

Indeed, labelled homomorphic encryption provides a new solution for these scenarios, where knowledge of the computation being carried out on the Client side can be used to boost the performance of the entire system (whilst still keeping Client-side computation at a very low cost).

Benchmarks We measured the timings of running homomorphic computation and decryption for the mean and covariance functions described in Section 6. We considered datasets with n rows and d columns where n and d are chosen among the following: $n \in \{2^{10}, 2^{12}, 2^{14}, 2^{16}, 2^{18}, 2^{20}\}$, $d \in \{1, 2, 4, 8, 12, 16, 20, 24\}$.

When computing only degree-1 functions, as is the case of the Means computation, our scheme offers two possible methods for homomorphic evaluation and decryption that permit avoiding pre-processing on the Client side, at a small extra cost in overall execution time. In Table 2 we show the timings obtained for datasets of fixed dimension $d = 2$ and varying number of rows, including the two options. We can see from the table the benefit of precomputation on the Client side: by allowing the Client to precompute some meta-information that depends on computation being carried out and on the labels associated with the input data, the overall efficiency of the protocol is significantly improved; moreover, the cost to the Client grows very slowly with the size of the computation.

Table 2. Timings (in seconds) for *Mean* query on datasets of dimension $d = 2$ and varying number of rows.

	With offline decryption					No offline decryption		
Dataset size	Eval	Dec (off)	Dec (on)	Dec (total)	Eval + Dec	Eval	Dec	Eval + Dec
1024	0.0001	0.0012	0.00001	0.0012	0.0012	0.0051	0.0070	0.0121
4096	0.0002	0.0046	0.00001	0.0046	0.0048	0.0205	0.0070	0.0275
16384	0.0008	0.0184	0.00001	0.0184	0.0192	0.0856	0.0070	0.0926
65536	0.0037	0.0739	0.00001	0.0739	0.0776	0.3249	0.0067	0.3316
262144	0.0122	0.2950	0.00001	0.2950	0.3072	1.3086	0.0063	1.3149
1048576	0.0630	1.1775	0.00001	1.1776	1.2405	5.5291	0.0063	5.5354

In Tables 3 (resp. 4) we provide timings for the processing of the Covariance query for a fixed number of 2 columns and growing number of rows (resp. for a fixed number of 4096 rows and a growing number of columns). Again, we can see that, although the decryption cost depends on the computation that was carried out, it remains very low in practice (e.g., 5 seconds for a dataset of over 2 million elements). More importantly, this offline decryption cost is *always* (significantly) dominated from above by the cost of running Eval on the server side. This means that the receiver could run the offline decryption while the server is computing the result, in which case the real decryption time becomes virtually constant and independent of f. Furthermore, the overall processing power required to run the protocol is much smaller than that required by solutions based on other somewhat homomorphic encryption schemes, as discussed in Appendix A.

Regarding bandwidth and storage needs, the 560Mbytes value we provided above for a data set of size 2^{20} and dimension 2 corresponds to the larger data set in Table 2, and it increases linearly

Table 3. Timings (in seconds) for *Covariance* queries on datasets of dimension $d = 2$ and varying number of rows.

Set size	Eval	Dec (off)	Dec (on)	Dec (tot)
1024	2.27730	0.00462	0.01046	0.01507
4096	8.98019	0.01845	0.01045	0.02891
16384	37.63178	0.07351	0.01046	0.08398
65536	142.28464	0.29512	0.00949	0.30460
262144	572.39846	1.17653	0.00949	1.18602
1048576	2391.64744	4.71030	0.00949	4.71979

with the number of elements in the data set. Recall that every ciphertext can be encoded into 272 bytes, so that the cost for the Cloud is $n \cdot d \cdot 272$ bytes. The comparison we present in Appendix A indicates that for the alternative somewhat homomorphic encryption scheme we analysed, the same data set would occupy 249 GBytes. The communication cost for every result sent from the Cloud to the Client is just 272 bytes.

Table 4. Timings (in seconds) for *Covariance* queries with $n = 4096$ rows and varying dimension d.

Set dim.	Eval	Dec (off)	Dec (on)	Dec (tot)
1	2.9804	0.0069	0.0032	0.0101
2	9.1942	0.0184	0.0104	0.0289
4	31.0953	0.0553	0.0349	0.0901
8	109.3363	0.1839	0.1139	0.2978
16	231.0709	0.3862	0.2487	0.6350
20	404.4647	0.6627	0.4737	1.1363
24	624.9429	1.0124	0.6621	1.6745

7.2 Application #2: GAS

Genetic Association Studies (GAS) look for statistically relevant features across the human genome, singling out those scientifically correlated with a given trait. GAS are carried out by performing a series of tests, each of them targetting a particular trait, and taking into consideration trait-associated information encoded in specific positions of an individual's genome, the so-called Single Nucleotide Polymorphisms (SNP).

Technically, SNPs are first identified by looking at statistically meaningful population groups and searching for points (single nucleotides) in the genome, where deviations can be used to identify outliers (i.e., most of the population is consistent at that position, but some individuals are not). The resulting SNP data-sets are then stored and data-mined for correlations, e.g., with medical information for the same population, to look for associations between SNP information and clinically relevant traits. The resulting correlations can then be used to define association tests to evaluate the likelihood of a (new) individual from the same population displaying the same trait.

Each test computes a Genetic Risk Score (GRS) that is essentially a weighted sum of the information collected for each SNP; the weights correspond to risk estimates computed for a reference population, typically the logarithm of an odds-ratio [32]. There are many applications of SNP genotyping [37], including personalised medicine, predictive medicine, forensics, prognosis, and many other potentially revolutionary (and not at all consensual) developments in medicine and beyond. Access to such tests is, for the most part, controlled by the health services of different countries, but a new trend of Direct-to-Consumer (DTC) genomic analysis is arising, where companies offer a multitude of association tests to the public. Privacy is obviously a paramount concern in such

services, as revealing genome information to any public or private organization can have unpredictable consequences, not only now but, perhaps most importantly, in the future, at a time when technology (and society itself) may have evolved to make use of genetic tests for purposes such as insurance, ethnicity profiling, etc.

In this paper we propose a technology that permits performing GAS securely, without compromising the privacy of the patient, and offering a trade-off between security and efficiency that makes it an ideal candidate for practical deployment. Our solution also protects potential intellectual property that the laboratories or medical centers carrying out the tests may want to protect, and it may be used in a commodity cloud, since all computation is carried out over encrypted data.

The computations of GAS From a computational point of view, the statistically relevance of an SNP for a particular individual can be represented as a pair (p, s), where $p \in \mathbb{N}$ is the position in the genome that identifies the SNP, and $s \in \{0, 1, 2\}$ is the distinguishing feature (possibly none) that is observable at that point in a particular individual. One of the inputs to a GAS is therefore a list G of such pairs. We will refer to $G[p]$ as the value of s that corresponds to position p in G. For each test included in the GAS, one needs to consider another input P, which is provided by the entity performing the study, and that contains the parameters required to perform a statistical computation over G. The test result, a Genetic Risk Score, is obtained by computing a floating point value $T(G, P)$, which is a weighted sum that can be written as [32]:

$$T(G, P) := \sum_{i=1}^{k} \left(a_i \cdot G[p_i]^2 + b_i \cdot G[p_i] \right) + c\,. \tag{1}$$

Here, $P = (k, c, [(p_i, a_i, b_i)]_{i=1}^{k})$, where k is the number of relevant SNPs, c is a floating point number, and (p_i, a_i, b_i) for $i \in [1..k]$ are the positions and association parameters (also floating point numbers) for each relevant SNP.

The total number of SNPs that have been documented up to date in the human genome is in the range of 150M. However, only a very small fraction of those, under 100K, has been looked at from a clinical analysis point of view[7] and, indeed, the number of medical conditions that have been scientifically related to a Genetic Risk Score is around 5000.[8] Furthermore, specific association tests, e.g., for a medical condition, will focus on a very small number of SNPs ranging from 1 or 2, to at most a few hundred and a safe estimate is that, over all current association tests, each of them will on average look at 50 SNPs. This places the number of clinically relevant SNPs, at present, at around 30K. Put differently, this is roughly the number of SNPs that one needs to look at in order to evaluate all the Genetic Risk Scores that have currently been associated with a medical condition.

We note that this stands in contrast with the current capability of extracting and data-mining SNP genotyping information: in Exome-Wide Association Studies [30] a given exome will yield roughly 50K potentially interesting SNPs; whereas in Genome-Wide Association Studies, these figures can go up to 10M. These figures indicate that the 30K upper-bound we identified above is likely to grow rapidly in coming years, and will likely increase by one order of magnitude in the relative short term, and potentially by two orders of magnitude in the long run. We will use these rough estimates to measure the practicality of our proposed solution for Privacy-Preserving GAS later on.

[7] `https://www.ncbi.nlm.nih.gov/snp`
[8] `http://www.disgenet.org/web/DisGeNET`

Security Model To describe the security goals of the system we propose, we present in Figure 2 the architecture of a Secure Direct-to-Consumer GAS.

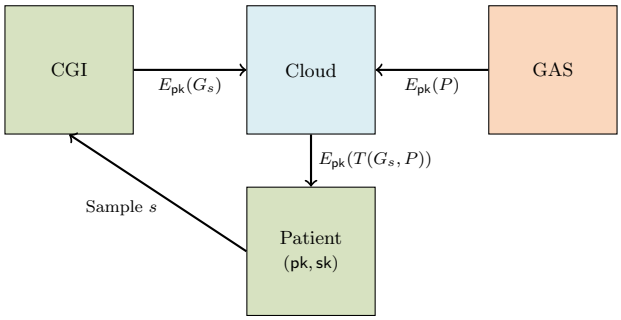

Fig. 2. Architecture of a Secure Direct-to-Consumer GAS.

The colors in the figure represent trust domains. The Patient wishes to be tested by the GAS service and trusts a Certified Genotyping Institution (CGI) to analyse a biological sample s, extract SNP information G_s, correctly encrypt it under the Patient's public key pk, and then erase all of the SNP-related information. This level of trust is implicit in GAS systems and cannot be eliminated from such a system, unless the Patient can perform the genotyping activities autonomously.

The GAS is trusted by the Patient to correctly encrypt the test parameters P. In security terms, the GAS is assumed to be honest-but-curious, in the sense that it is trusted to follow a set of rules of the protocol, but not trusted to learn the genetic data of the Patient—even if it could learn the information that the Patient provided to the Cloud.

The Cloud is also assumed to be honest-but-curious by both the Patient and the GAS. More precisely, the following behaviour is assumed for the Cloud:

- The Cloud is trusted by the GAS not to reveal the encrypted test parameters to the Patient;
- The Cloud is trusted by the Client to correctly perform the computation (over encrypted data).[9]

Note that the Cloud is *not* trusted by the Client to learn genetic data, and it may also be assumed to share data with the GAS, which means that this trust model is compatible with the most likely scenario that the GAS owns or contracts the Cloud service itself, and uses it to provide a service to the Patient.

Finally, we assume that there is a predefined set $\mathcal{L}$ of *all* positions (*loci*) of relevant SNPs, which is public and known by all parties. This could be the union of all positions that the GAS may test in all of its analyses—if this is not sensitive information from the point of view of the GAS—or it may be a larger set of all positions of SNPs that are known to be clinically relevant by the scientific community. In the first case we would have $|\mathcal{L}|$ in the range of a few hundred, and in the second case we would have $|\mathcal{L}|$ in the range of the 30K, as things stand today [27,14]. These numbers may increase by a factor of 10 and even 100 in the short and long terms, respectively. This way, in order to ask for a test that involves only a subset $\mathcal{L}' \subset \mathcal{L}$ of positions, the GAS can simply work with the full set $\mathcal{L}$ and set $a_i = b_i = 0$ for all $p_i \in \mathcal{L} \setminus \mathcal{L}'$.
Under these assumptions, our solution guarantees that *nothing is leaked* about the genetic information of the Patient *nor* about the concrete parameters used by the GAS to perform its tests.

[9] We note that recent developments in commodity hardware such as Intel's Software Guard Extensions permit replacing this assumption with an attestation mechanism [3].

Furthermore, if one sets $\mathcal{L}$ to include all clinically relevant SNPs, then no-one except the Patient and the medical center defining the tests will learn which traits are being tested—crucially this means that all access patterns over the stored genome data are kept private. Otherwise, it will be publicly known that the Patient was tested at positions relevant for a specific GAS and no more.

Our solution We propose to use multi-user Labeled Homomorphic Encryption scheme introduced in this paper to carry out the computation over encrypted data in the Cloud. The natural question that arises is whether such a solution is practical. For this reason, we will also provide an extensive experimental validation that answers this question in the affirmative. We begin by describing our protocol:

SETUP: The Patient generates a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$ for the mu-labHE scheme, and publishes its public key.
SNP ENCRYPTION: The CGI generates G_s, containing a pair (p_i, s_i) for all positions $p_i \in \mathcal{L}$. It then generates a key pair $(\mathsf{usk}_{CGI}, \mathsf{upk}_{CGI}) \leftarrow \mathsf{KeyGen}(\mathsf{pk})$, and constructs a list EG_s of tuples $\langle p_i, \mathsf{Enc}(\mathsf{pk}, \mathsf{usk}_{CGI}, 2 \cdot i - 2, s_i^2), \mathsf{Enc}(\mathsf{pk}, \mathsf{usk}_{CGI}, 2 \cdot i - 1, s_i) \rangle$, for $p_1, \ldots, p_{|\mathcal{L}|} \in \mathcal{L}$, and sends $(EG_s, \mathsf{upk}_{CGI})$ to the Cloud.
PARAMETER ENCRYPTION: For each test with parameters P that it may wish to perform, GAS generates a key pair $(\mathsf{usk}_{GAS}, \mathsf{upk}_{GAS}) \leftarrow \mathsf{KeyGen}(\mathsf{pk})$, constructs a list EP of tuples $\langle p_i, \mathsf{Enc}(\mathsf{pk}, \mathsf{usk}_{GAS}, p_i, 2 \cdot i - 2, a_i), \mathsf{Enc}(\mathsf{pk}, \mathsf{usk}_{GAS}, 2 \cdot i - 1, b_i)) \rangle$, for $p_1, \ldots, p_{|\mathcal{L}|} \in \mathcal{L}$, and sends $(EP, \mathsf{Enc}(\mathsf{pk}, \mathsf{usk}_{GAS}, 2 \cdot |\mathcal{L}|, c), \mathsf{upk}_{GAS})$ to the Cloud.
TEST COMPUTATION: The Cloud uses the Eval algorithm of mu-labHE to compute an encryption $E_{\mathsf{pk}}(T(G_s, P))$ and sends it back to the Patient along with $(\mathsf{upk}_{CGI}, \mathsf{upk}_{GAS})$.
RESULT: The patient uses its secret key sk to recover the test result.

How the test result is used by the Patient is beyond the scope of this paper. We envision that for some tests the patient may be provided sufficient information to analyse the result autonomously; for other tests, the result can be shared with the GAS after the protocol is concluded to obtain an informed analysis of the implications.

Security rationale We have proven that our encryption scheme has semantic security and context-hiding, and that it can evaluate computations that can be expressed as quadratic polynomials. Semantic security ensures that honest-but-curious adversaries, such as those contemplated in our model, obtain no information about encrypted data, except for its length. Context hiding ensures that even the Patient, with knowledge of his secret key, obtains no information about the (possibly proprietary) test parameters P provided by the GAS. To show that we achieve our security goal, it therefore suffices to ensure that the length of the encrypted data exchanged by the parties does not reveal sensitive information. On the other hand, to show that we achieve correctness, it suffices to show that the GAS computation can be correctly expressed as a quadratic polynomial on the values that are provided to the Cloud in encrypted form.

For security, it is easy to see that the length of all data provided to the Cloud is independent of both Patient data and test parameter data. We achieve this by fixing the set $\mathcal{L}$ of relevant positions and always encoding information in *all* positions for all patients and all tests. Indeed, both EG_s and P will always be of length $2 * |\mathcal{L}|$. The lengths of the encrypted values themselves, i.e., SNP features and floating-point test parameters, are fixed by their computational representations, e.g., 64-bits for integers and double-precision floating point numbers will suffice.

Although this approach may seem wasteful of resources, we emphasize that this is essential to ensuring that the Cloud (or some external observer) can infer *nothing* from an encrypted version

of G_s and P, in addition to the public set $\mathcal{L}$ itself, under the assumption that the encryption scheme is semantically secure. Furthermore, as we will see in our experimental evaluation, the efficiency of our homomorphic encryption scheme works as an enabler for this level of security, as it permits performing computations in reasonable time. Previous solutions aiming for this level of security either relied on heavy cryptographic machinery such as oblivious RAM [29]—to hide all access patterns to the stored genomic data—*and* were significantly heavier from a computational and infrastructural point of view because computation was only feasible if carried out by multiple collaborating (and assumed to be non-colluding) servers—our solution works with a single Cloud server. More efficient solutions proposed in the literature either compromise on security or they require dedicated trusted hardware on the Patient side [17,4].

It remains to show that our protocol is correct. This follows from the fact that $T(G_s, P)$ can be computed as a quadratic polynomial of the form $x_0 \cdot y_0 + x_1 \cdot y_1 + \ldots + x_k \cdot y_k + c$, where $k = 2 \cdot |\mathcal{L}| - 1$ and, for $i \in [0..|\mathcal{L}|)$, we have $x_{2 \cdot i} = G[p_{i+1}]^2$, $x_{2 \cdot i+1} = G[p_{i+1}]$, $y_{2 \cdot i} = a_{i+1}$ and $y_{2 \cdot i+1} = b_{i+1}$.

Benchmarks Figure 3 shows the timing data we collected when evaluating our protocol on data sets of increasing sizes. The offline encryption and decryption times increase linearly with the number of SNPs, although the offline decryption time is under 90ms even for 30000 SNPs, whereas the off-line encryption time gradually grows up to 45s. This difference is justified by the fact that off-line decryption precomputation works on raw label data, whereas off-line encryption is creating a large number of ciphertexts. For this reason we have shown the two curves in logarithmic scale. This also shows that the overall decryption time, even accounting for the preprocessing is very light: note that on-line decryption takes constant time in the range of 3ms. Online encryption time, on the other hand is very fast, and can be done in under 24ms even for 30000 SNPs. Finally, the homomorphic computation in the cloud, grows linearly with the number of points, and it is reasonably small, clearly in the range of practicality, and even using a single modest server and no parallelism. In our machine, the processing time was around 47s for a risk analysis involving 30000 SNPs. We recall that this was the estimated worst case scenario for the union of SNPs corresponding to all GAS-relevant information known to today. The size of the encrypted data processed by the cloud is, in this case, 32MByte, half of it produced by the Patient and the rest by the medical centre. For each test that the Medical Centre might perform, an additional 16MByte would be uploaded to the Cloud, and could be used over the same genomic data provided by the Patient.

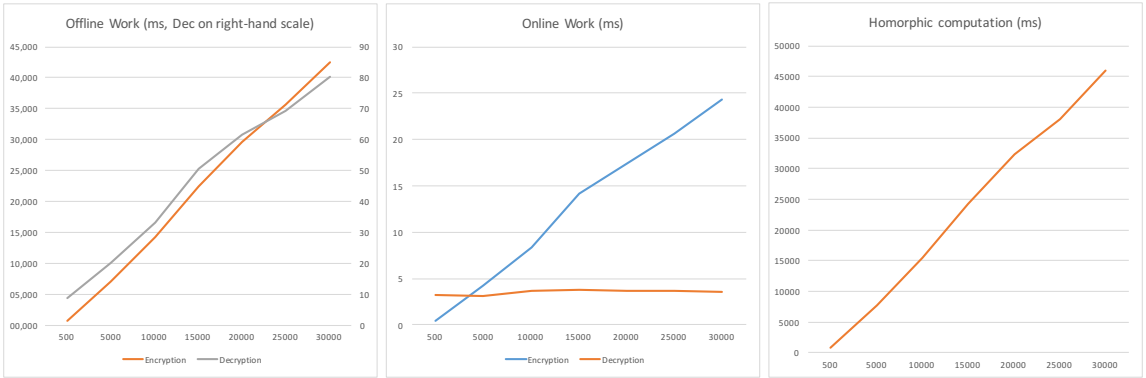

Fig. 3. Timings for the various algorithms in the secure GAS protocol for increasing numbers of SNPs.

To evaluate the scalability of our solution, we considered a Map-Reduce scenario, in which the multiplicative part of the weighted sum computation for GAS is split by multiple servers in the Cloud, in such a way that many partial weighted sums can be computed in parallel, and later on combined into a final aggregated result. Table 5 shows the result of our analysis. Presented times include the overhead of reading data from storage and writing the results back for two scenarios where one would outsource batches of 100K and 50K SNPs for partial sum computation in each server. One can see that the time for partial sum computation is a fraction of that required for the product computation and, similarly, the aggregation time for 10 parallel jobs is under one millisecond. We are, of course ignoring in our analysis the communication time and, to compensate for that, we include also in the amount of data that must be conveyed to each server and the size of the results that must be retrieved. Overall, using this strategy, to compute a GAS over 1M SNPs using 10 servers, the overall computation time would be in the range of 3 minutes, excluding the communication overhead; this would remain essentially constant for 2M SNPs if one can duplicate the number of servers.

Table 5. Timings for scalable secure GAS using map-reduce.

SNPs per server	**50K**	**100K**	**Unit**
Product time per server	82	168	sec.
Partial sum time per server	2,52	5.13	sec.
Sum time (10 servers)	0,8	0.9	millisec.
Raw data per server	53.1	106.3	Mbytes
Result data per server	256	256	bytes

8 Further related work

Ayday, Raisaro, Hubaux, and Rougemont [2] presented a cryptographic protocol for a security model that inspired our solution for secure GAS. The roles of the Patient and the Certified Institution are the same, but the trust models between the Cloud (there called Storage and Processing Unit) are problematic for real-world applications. Indeed, the Cloud and the Medical Centre are assumed to be able to know which and how many SNPs are being tested in a particular evaluation, which means that both the privacy of the Patient and that of the Medical Centre can be damaged be engaging in the computation. Nevertheless, the solution guarantees that the concrete SNP values are known only to the Patient, and that the test parameters used by the Medical Centre are not revealed. This is achieved by revealing the test result to the Medical Centre implicitly, whereas in our solution the Patient can choose to do so.

Danezis and De Cristofaro [17] revisit the scenario above and propose a solution that no longer leaks which and how many SNPs are being tested. This is achieved by shifting part of the computation to the Client side and requires the Patient to possess a trusted hardware token (e.g., a smartcard) to participate in the protocol. This means that the scalability properties and deployment ease of this solution are limited. The same authors also present an alternative solution that is based on secret-sharing-based multiparty computation, but here the trust model is different: one must assume that the Cloud and the Medical Centre do not collude.

Barman et al. [4] look at various potential security problems in Genetic Risk Tests such as the ones we consider in this paper. In particular, the approach of encrypting the entire set of relevant

SNPs, as we propose, is identified as a valid countermeasure for *test inference attacks* enabled by the SNP position leaks described above. The drawback pointed out for this countermeasure is impact on performance and scalability. The proposed solution in that paper implies a (controlled) compromise of the Medical Centres' privacy. Our work shows that using present-day Cloud technology, performance and scalability need not be an issue, and that such a compromise may not be needed.

More intricate solutions for hiding SNP patterns have been proposed based on Oblivious RAM [29] and Private Information Retrieval [26]. However, here the permformance and scalability problems associated with existing solutions for these cryptographic problems are significant and, indeed, as pointed out by [4] the functionalities supported by such primitives are an overkill for this application scenario. We have discussed the main differences to our solution in the main text.

Other homomorphic encryption schemes could be used to address the same scenario of genomic analysis we consider. Dowlin et al. [33] present a framework for deploying homomorphic encryption-based secure computation for bioinformatics. The authors describe how the Simple Encrypted Arithmetic Library (SEAL) can be used to compute various bioinformatics-related statistics using the YASHE [7] homomorphic encryption scheme (although SEAL 2.0 implements the FV homomorphic encryption scheme). Overall, we found that implementing computations such as the ones adopted in this paper with SEAL, at a security level of 128-bits, turns out to be $50\times - 400\times$ slower and require $400\times$ more bandwidth than using the solution we propose.

A series of other works focuses on using somewhat homomorphic encryption on bioinformatics applications. We mention a few. Kim and Lauter [31] present a study on how various genomic analyis statistics can be carried out using both the YASHE [7] and BGV [22] schemes. In [6], predictive analysis is carried out over encrypted health data, and in [38], HELib [25] is used to construct a secure solution to the challenges of the 2015 iDASH challenge.

9 Conclusions

We presented a new methodology for processing remotely outsourced data in a privacy preserving way via the notion of Labeled Homomorphic Encryption. We showed an efficient realization and implementation of this primitive that targets computations described by degree-2 polynomials, with applications to executing statistical functions on encrypted data. Our experiments confirmed the practicality of our solution showing that it outperforms solutions based on somewhat homomorphic encryption. Our current solutions achieve privacy against a honest-but-curious Cloud server. In order to achieve security against malicious servers, one can use verifiable computation protocols in a generic fashion, as explained in [19]. Unfortunately, applying this idea generically to our schemes does not yield an efficient solution. Informally this is because modeling algebraic operations over $\mathbb{Z}_N^*$ is expensive when using state-of-the-art VC protocols (such as [35]). Designing an ad-hoc verifiable computation mechanism for our schemes while preserving efficiency is therefore a promising future direction for this work.

References

1. Gail-Joon Ahn and Anupam Datta, editors. *Proceedings of the 13th Workshop on Privacy in the Electronic Society, WPES 2014, Scottsdale, AZ, USA, November 3, 2014*. ACM, 2014.
2. Erman Ayday, Jean Louis Raisaro, Jean-Pierre Hubaux, and Jacques Rougemont. Protecting and evaluating genomic privacy in medical tests and personalized medicine. In Ahmad-Reza Sadeghi and Sara Foresti, editors, *Proceedings of the 12th annual ACM Workshop on Privacy in the Electronic Society, WPES 2013, Berlin, Germany, November 4, 2013*, pages 95–106. ACM, 2013.

3. Manuel Barbosa, Bernardo Portela, Guillaume Scerri, and Bogdan Warinschi. Foundations of hardware-based attested computation and application to SGX. In *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*, pages 245–260. IEEE, 2016.
4. Ludovic Barman, Mohammed Taha Elgraini, Jean Louis Raisaro, Jean-Pierre Hubaux, and Erman Ayday. Privacy threats and practical solutions for genetic risk tests. In *2015 IEEE Symposium on Security and Privacy Workshops, SPW 2015, San Jose, CA, USA, May 21-22, 2015*, pages 27–31. IEEE Computer Society, 2015.
5. Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In Sushil Jajodia and Javier López, editors, *ESORICS 2008*, volume 5283 of *LNCS*, pages 192–206. Springer, October 2008.
6. Joppe W. Bos, Kristin Lauter, and Michael Naehrig. Private predictive analysis on encrypted medical data. Cryptology ePrint Archive, Report 2014/336, 2014. `http://eprint.iacr.org/2014/336`.
7. Joppe W. Bos, Kristin E. Lauter, Jake Loftus, and Michael Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In Martijn Stam, editor, *Cryptography and Coding - 14th IMA International Conference, IMACC 2013, Oxford, UK, December 17-19, 2013. Proceedings*, volume 8308 of *Lecture Notes in Computer Science*, pages 45–64. Springer, 2013.
8. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012*, pages 309–325. ACM, January 2012.
9. Emmanuel Bresson, Dario Catalano, and David Pointcheval. A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In Chi-Sung Laih, editor, *ASIACRYPT 2003*, volume 2894 of *LNCS*, pages 37–54. Springer, November / December 2003.
10. Dario Catalano and Dario Fiore. Practical homomorphic MACs for arithmetic circuits. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 336–352. Springer, May 2013.
11. Dario Catalano and Dario Fiore. Using linearly-homomorphic encryption to evaluate degree-2 functions on encrypted data. In *ACM CCS 2015 – 22nd ACM Conference on Computer and Communication Security*, pages 1518–1529, 2015.
12. Dario Catalano, Dario Fiore, and Bogdan Warinschi. Homomorphic signatures with efficient verification for polynomial functions. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 371–389. Springer, August 2014.
13. Anamaria Costache, Nigel P. Smart, Srinivas Vivek, and A. Waller. Fixed point arithmetic in SHE scheme. *IACR Cryptology ePrint Archive*, 2016:250, 2016.
14. Loredana Covolo, Sara Rubinelli, Elisabetta Ceretti, and Umberto Gelatti. Internet-based direct-to-consumer genetic testing: A systematic review. *Journal of Medical Internet Research*, 17(12):e279, 12 2015.
15. Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013*, volume 8134 of *LNCS*, pages 1–18. Springer, September 2013.
16. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, August 2012.
17. George Danezis and Emiliano De Cristofaro. Fast and private genomic testing for disease susceptibility. In Ahn and Datta [1], pages 31–34.
18. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. `http://eprint.iacr.org/2012/144`.
19. Dario Fiore, Rosario Gennaro, and Valerio Pastro. Efficiently verifiable computation on encrypted data. In *ACM CCS 14*, pages 844–855. ACM Press, 2014.
20. Rosario Gennaro and Daniel Wichs. Fully homomorphic message authenticators. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 301–320. Springer, December 2013.
21. Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.
22. Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 850–867. Springer, 2012.
23. Shafi Goldwasser and Silvio Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, pages 365–377, New York, NY, USA, 1982. ACM.
24. Shai Halevi and Victor Shoup. Helib. https://github.com/shaih/HElib.

25. Shai Halevi and Victor Shoup. Algorithms in helib. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 554–571. Springer, 2014.
26. Yizhou Huang and Ian Goldberg. Outsourced private information retrieval. In *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society*, WPES '13, pages 119–130, New York, NY, USA, 2013. ACM.
27. Andrew D Johnson, Anupama Bhimavarapu, Emelia J Benjamin, Caroline Fox, Daniel Levy, Gail P Jarvik, and Christopher J O'Donnell. Clia-tested genetic variants on commercial snp arrays: Potential for incidental findings in genome-wide association studies. *Genetics in medicine : official journal of the American College of Medical Genetics*, 12(6):355–363, 06 2010.
28. Marc Joye and Benoît Libert. Efficient cryptosystems from 2^k-th power residue symbols. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 76–92. Springer, May 2013.
29. Nikolaos P. Karvelas, Andreas Peter, Stefan Katzenbeisser, Erik Tews, and Kay Hamacher. Privacy-preserving whole genome sequence processing through proxy-aided ORAM. In Ahn and Datta [1], pages 1–10.
30. Thorsten Kessler, Baiba Vilne, and Heribert Schunkert. The impact of genome-wide association studies on the pathophysiology and therapy of cardiovascular disease. *EMBO Molecular Medicine*, 8(7):688–701, 2016.
31. Miran Kim and Kristin Lauter. Private genome analysis through homomorphic encryption. Cryptology ePrint Archive, Report 2015/965, 2015. `http://eprint.iacr.org/2015/965`.
32. Bo Eskerod Madsen and Sharon R. Browning. A groupwise association test for rare mutations using a weighted sum statistic. *PLoS Genet*, 5(2):1–11, 02 2009.
33. Kim Laine Kristin Lauter Michael Naehrig John Wernsing Nathan Dowlin, Ran Gilad-Bachrach. Manual for using homomorphic encryption for bioinformatics. Technical report, November 2015.
34. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, May 1999.
35. Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society Press, May 2013.
36. R L Rivest, L Adleman, and M L Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation, Academia Press*, pages 169–179, 1978.
37. Wikipedia. Single-nucleotide polymorphism — wikipedia, the free encyclopedia, 2016. [Online; accessed 21-October-2016].
38. Yuchen Zhang, Wenrui Dai, Xiaoqian Jiang, Hongkai Xiong, and Shuang Wang. Foresee: Fully outsourced secure genome study based on homomorphic encryption. *BMC Medical Informatics and Decision Making*, 15(5):S5, 2015.

A Comparison to SHE

As we mentioned, a solution for the applications considered in our work can also be obtained by using somewhat homomorphic encryption schemes supporting quadratic polynomials. As we argue in this section, these solutions are however more expensive in terms of both communication and computation.

Specifically, we compared our solution against the scale-invariant FV [18] homomorphic encryption scheme and its implementation in the library SEAL 2.0 [33].[10] We considered an instantiation of FV with parameters optimized to support the same security level and the same computations as our solution, namely degree-2 polynomials over integers with 128-bits precision (considering that both our applications aim to support up to 2^{20} additions of degree-2 terms). Integers were encoded in FV using the `IntegerEncoding` utility provided by SEAL. The FV scheme works over a polynomial ring $R := \mathbb{Z}[X]/(X^n + 1)$, so that its message space is $R_t := R/tR$ and its ciphertext space is $R_q := R/qR$, for two integers t and q. In our instantiation we used $n = 4096$, $t = 2^{26}$, and q a 116-bits prime. In our experimentation we also tested the BGV homomorphic encryption [8] implemented in HElib [24]. However, for computations with few multiplications, such as the ones in our applications, a scale-invariant scheme such as FV performs better.

[10] `http://sealcrypto.codeplex.com`

Table 6. Timings (s) for *Mean* and *Covariance* queries on data sets of dimension $d = 2$ and varying number of rows using FV.

Set size	Eval(Mean)	Eval(Cov)	Dec(Mean)	Dec(Cov)
1024	0.4695	122.3210	0.0127	0.0337
4096	1.9499	486.0580	0.0140	0.0311
16384	8.7016	1931.2000	0.0127	0.0311

Micro-benchmarks We first discuss the communication/storage costs. Every ciphertext of the FV scheme instantiated with the above parameters is 118.8 KBytes. If we consider a dataset with $n = 2^{20}$ rows and $d = 2$ columns, it means that the Cloud has to store about 249 GBytes, whereas with our solution the storage cost would be about 560 MBytes. Timing-wise, the basic operations took 129.243ms for key generation, 30.284ms for encryption and 9.869ms for decryption, so a parallel can be drawn to that in Table 1. We can see that, although our key generation time is slightly larger (this is justified because we precompute some constant values that are useful to speed-up encryption, evaluation and decryption), the two values are close. On the other hand, an encryption in our scheme is over 80 times faster and decryption time for a fresh ciphertext is at least three times faster. Nevertheless, as we will see below, the most significant differences arise in the evaluation of computations over programs of increasing size.

A.1 Application #1: Secure statistics

We measured the performance of the FV somewhat homomorphic encryption in processing the statistics queries of our first application. We measured the timing of homomorphic computation and decryption for the mean and covariance functions described in Section 6. Since here we are only interested in showing a comparison to our solution we only report the timings obtained when considering datasets of fixed dimension $d = 2$ and varying number of rows $n \in \{2^{10}, 2^{12}, 2^{14}\}$. We note that for larger datasets the FV-based solution we were not able to run the benchmarks in the same machine where we collected the remaining data due to excessive use of memory. The timings are provided in Table 6. By comparing the numbers against those obtained with our solution provided in Tables 2 and 3, one can see that our solution is considerably faster. The processing (including evaluation at the Cloud and decryption at the Client) of a Mean query is at least $400\times$ faster, while that of a Covariance query is at least $50\times$ faster.

A.2 Application #2: GAS

We measured the performance of using FV in the GAS application. Considering a risk analysis involving 30K SNPs, the size of encrypted data processed by the Cloud is, using FV, 14 GBytes, which is over 400 times more than the space required by our solution. For the same risk analysis, the processing time for computing the Genetic Risk Score using FV was around 100 times slower than the ones reported in Section 7. The benchmarking platform quickly becomes unstable due to the large memory requirements.