Private Predictive Analysis on Encrypted Medical Data

Joppe W. Bos, Kristin Lauter, and Michael Naehrig

Cryptography Research Group, Microsoft Research, Redmond, USA

Abstract. Increasingly, confidential medical records are being stored in data centers hosted by hospitals or large companies. As sophisticated algorithms for predictive analysis on medical data continue to be developed, it is likely that, in the future, more and more computation will be done on private patient data. While encryption provides a tool for assuring the privacy of medical information, it limits the functionality for operating on such data. Conventional encryption methods used today provide only very restricted possibilities or none at all to operate on encrypted data without decrypting it first. Homomorphic encryption provides a tool for handling such computations on encrypted data, without decrypting the data, and without even needing the decryption key.

In this paper, we discuss possible application scenarios for homomorphic encryption in order to ensure privacy of sensitive medical data. We describe how to privately conduct predictive analysis tasks on encrypted data using homomorphic encryption. As a proof of concept, we present a working implementation of a prediction service running in the cloud (hosted on Microsoft's Windows Azure), which takes as input private encrypted health data, and returns the probability for suffering cardiovascular disease in encrypted form. Since the cloud service uses homomorphic encryption, it makes this prediction while handling only encrypted data, learning nothing about the submitted confidential medical data.

1 Introduction

More and more businesses and individuals confide their data to cloud services and outsource computational tasks on their data to third-party service providers. This raises concerns about the privacy of sensitive information since data is stored in external, off-premise data centers. In particular in the health sector, sensitive personal patient records need to be kept confidential. Privacy of sensitive information can be guaranteed, if it is encrypted by the data owner before being uploaded to a cloud service. In that way, only the legitimate data owner can access the data by decrypting it using their private decryption key. But encryption limits the possibility to outsource computation on the externally stored information, especially if the data center does not have access to the decryption key, since the key, for standard encryption schemes, is needed to decrypt the data so as to perform a computation upon it. For example, a very specific task like searching an encrypted database, without decrypting all of its entries first, requires special types of encryption schemes with large computational overhead, and even a simple statistical analysis becomes impossible with standard methods of encryption.

However, exactly such computational tasks are often crucial to the business value of maintaining databases of customer or patient information. For example, a hospital may want to be evaluated on its performance on the basis of its patients' health records, but might not want to disclose the details of all patient records. In another example, a patient may want to use a web service that stores and maintains all her medical records in a centralized place, but she might not trust the cloud service to keep her private health data confidential. Still, she may want to obtain information about her health status such as a prediction of whether or not she will contract a specific disease.

With homomorphic encryption, many such scenarios can be realized, because a homomorphic encryption scheme allows computations on encrypted data without decrypting it. This

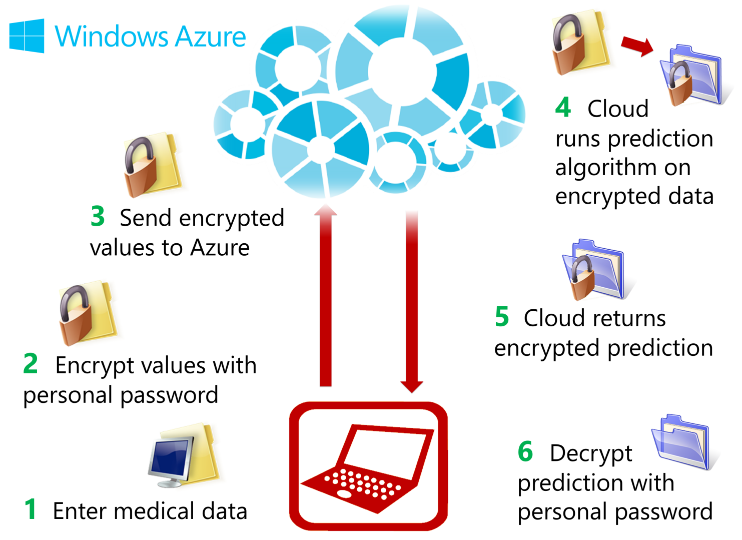
Windows Azure
4 Cloud runs prediction algorithm on encrypted data
3 Send encrypted values to Azure
5 Cloud returns encrypted prediction
2 Encrypt values with personal password
6 Decrypt prediction with personal password
1 Enter medical data

Fig. 1. Cloud service for privately predicting cardiovascular disease on encrypted medical data.

means that for example a cloud prediction service can predict the likelihood of contracting a disease by only working on an encrypted medical record. At no time during the computation is it required to decrypt the data, and the result is produced in encrypted form. Only after the patient receives the encrypted prediction result on a local, private device, she decrypts to learn the prediction. This means that the cloud service only sees data in encrypted form and never learns any information about the encrypted data values.

Contributions. We present a working implementation of a cloud service that demonstrates an application of outsourced prediction algorithms on encrypted, confidential medical data. The algorithm we implement predicts the chance of having a heart attack based on a few body measurements. The service consists of a client application on a private device and a cloud application that runs in the Microsoft Windows Azure cloud service. The client application collects user health data, encrypts it and sends the encrypted record to the cloud application, which runs the prediction algorithm on the encrypted record. The cloud produces an encrypted prediction result, which it sends back to the client application. The user can then decrypt and learn the likelihood of having a heart attack. See Figure 1 for a schematic overview of the implementation. The Magma code for implementing the homomorphic encryption schemes used here will be publicly released. A software demonstration is available as a live demo of the prediction service at the link provided at the end of this article.

In addition to the implementation of the homomorphic encryption scheme and the cloud service, the main contribution of this work is to give details of using a leveled homomorphic encryption scheme for real-life predictive analysis. The term *leveled* refers to the fact that the homomorphic encryption scheme cannot correctly and securely carry out an arbitrary computation; instead, the scheme can only be used to compute functions up to a certain complexity, or level, that is fixed in advance. Such a scheme is typically more efficient and practical than a scheme that allows arbitrary computation, in particular if the function to be computed has low multiplicative complexity. In this case, better performance can be achieved by tailoring the parameters of the scheme to the specific computation that is desired. But parameters must be set carefully to ensure correctness and security, and these parameters depend on the size of the function to be computed, the size of the inputs, and the method for encoding real data. The necessity of choosing the right parameters makes it difficult to implement leveled homomorphic encryption for practical applications. We propose practical methods for handling these aspects of homomorphic encryption schemes, including a parameter selection algorithm, to realize valuable functions occurring in predictive analysis, such as logistic regression and Cox proportional hazard regression. We also give references to indicate the extent to which such functions are prevalent in predictive analytics in health care, and an overview of scenarios for private computation where such functions are relevant.

2 Scenarios for Private Computation on Medical Data

In this section, we outline a number of scenarios in which private computation on patient medical data is desirable. Homomorphic encryption provides a potentially viable solution for some, but not all, of these scenarios.

Patient Records

Many commercial options exist today for storage and online access to patients' electronic medical records (EMR). In [1], patient controlled encryption (PCE) was proposed, allowing patients to outsource storage of their personal records in encrypted form. This proposal included the possibility to use a hierarchical key structure to enable sharing of keys for only selective branches of the record (e.g. for only dental records), and a proposal to use searchable encryption to enable the patient to privately and remotely search the record. In [2], the authors propose using attribute-based encryption to achieve a more flexible and dynamic way to share permissions on selective parts of an EMR. In [3], homomorphic encryption was proposed to allow a certain amount of computation on a patient's encrypted medical data, including for example, data which has been uploaded in encrypted form from various sources such as health monitors, labs or doctors. The encrypted result of the computation can be returned to the patient for decryption, or to any other designated party with whom the patient has shared decryption keys, such as a doctor or family member.

Medical Databases for Research

When a large amount of data from many different patients, relevant to a particular disease (such as cancer) is available, building predictive models from that data using machine learning can contribute substantially to medical science and the public good. Such models can help to identify both genetic and environmental risk factors and to form the basis for predictive analysis. However, privacy concerns, even when the data is anonymized, need to be weighed in the balance (see [4] for a recent study on re-identification of data in this context where certain high-profile individuals are re-identified). A recent news article in Science [5] juxtaposes different points of view on issues of genomic privacy and summarizes the approach presented here to protecting privacy while allowing computation. Different challenges arise and different solutions are possible depending on whether the data is publicly available for research or whether the data is privately compiled or controlled by a single hospital or company:

- *Publicly Available Databases.* This setting allows the greatest access to the data, thereby creating the steepest privacy challenge. Differential privacy-preserving techniques are potentially applicable. Differential privacy is implemented directly on unencrypted data, and works by providing approximate answers to queries to partially obscure exact information. See [6] for an overview of differential privacy techniques. It seems likely that differential privacy could be used to approximately answer queries based on a publicly known trained model, however, it is less clear how well differential privacy would succeed when building trained models with machine learning techniques. Differential privacy can be considered orthogonal to the solutions based on homomorphic encryption considered here, since it works on plaintext data. An outstanding research question is to design solutions which effectively combine differential privacy with homomorphic encryption in a useful way. Another possibility for achieving anonymity is to use a syntactic approach, which typically offers much higher data utility. The recent paper [7] discusses the advantages and limitations of syntactic approaches and differential privacy. For publicly available databases, aside from the above, recent advances in cryptographic research suggest other potential solutions for building trained models using secure multi-party computation (MPC) or multi-key homomorphic systems. In another direction, earlier recommendations were given in [8] for protecting privacy in population-based studies through the mechanisms of policy, consent, and regulation.

– *Privately Compiled Databases.* The following two scenarios are more suitable than the above scenario for homomorphic encryption. Firstly, private companies developing and deploying tools for genetic research amass data through their business. Depending on agreements with customers, they may be bound to maintain confidentiality of patient data when outsourcing computation on such databases. Secondly, hospitals or clinics also maintain patient records, but are bound by regulations and patient agreements to handle this data confidentially.

  In both of these settings, the hospital or company may seek to improve their business by learning from their data. A cloud service provider can offer storage and/or analysis on this privately held data, applying machine learning techniques to learn from the data to build predictive models. Homomorphic encryption allows the outsourcing of computation on the data, encrypted under the key of the hospital or company. In [9] it was demonstrated that basic machine learning algorithms such as the linear means classifier and Fisher's linear discriminant classifier can be performed efficiently to build encrypted models from encrypted data on a small scale, although efficiency degrades rapidly as the size of the database grows. So to summarize, an untrusted cloud service can provide both training of new models ([9]) and prediction based on known models (the present work) when given access to the data in homomorphically encrypted form.

Direct-to-Patient Services

This is the application which we demonstrate in the present paper. A cloud service may wish to offer genetic analysis or predictive services over the internet without compromising the consumer's privacy. Using homomorphic encryption, it is possible to offer confidential analysis or predictive services based on a learned predictive model. The consumer uploads the data in encrypted form, and the cloud service computes a result in encrypted form, which is then returned to the customer in encrypted form. The predictive model may be publicly known, as in the cardiovascular disease application we present here, or it may be a proprietary model, learned from privately held data, such as in the scenario labeled *Privately compiled databases* above. In the applications demonstrated here, it is assumed that the predictive model is publicly known. The next section describes a number of publicly known models which have been developed over time to predict the likelihood of various diseases.

Verifiability

Verifiability is an issue which is relevant for outsourced computation of any kind. Cryptographic schemes for verifying outsourced computation have been proposed and are just beginning to become practical ([10, 11]). However such schemes do not apply directly to homomorphically encrypted data in an efficient way, and much work remains to be done to effectively combine these techniques. In many scenarios we propose, however, a naive solution is reasonable: simply outsource computation to multiple servers and check locally that the outputs agree once the computations are returned and decrypted. This solution provides no guarantees against multiple colluding adversaries, but makes sense when a cloud service provider is modeled as a rational economic player who wants to build trust in the service it provides. In that case, outsourcing multiple copies and checking multiple computations for agreement simply serves as a partial defense against accidental errors, deletions, and denial of service attacks.

3 Models for Predictive Analysis in Health Care

Over the last 50 years, a number of powerful mathematical models have been developed, studied, and used to perform valuable predictive analysis in health care. In this section, we detail common models and explain how to implement them for use with homomorphic encryption. We focus on logistic regression and the Cox proportional hazard model as representative examples. Logistic regression has been commonly used to predict whether a patient will suffer from a particular disease, like cardiovascular disease (CVD) [12] or diabetes [13]. Logistic regression has also been proposed as a tool to predict the probability of survival in blunt trauma, and has been used as the basis for calculating the Trauma and Injury Severity Score (TRISS) for trauma audit [14].

In [15], numerous examples of medical uses of logistic regression in the study of cardiovascular disease and diabetes are noted, including testing gender as a predictor of mortality after certain types of heart surgery [16], correlating certain genotypes with the risk of cardiovascular disease [17], and relating certain protein abnormalities with occurrence of diabetes [18].

In the Framingham heart study [19], risk functions for CVD were calculated using various models, including linear regression, logistic regression, and Cox proportional hazard regression. Linear regression models were used to relate the risk of stroke to poor cognitive function [20]. First the discriminant model and logistic regression and then the Cox proportional hazard regression model were used to build the Framingham Risk functions, which are now widely available for predicting the likelihood of heart attacks [19, 12].

3.1 The Cox Proportional Hazard Model

The *Framingham heart study* [19] followed roughly 5,000 patients from Framingham, Massachusettes, for many decades starting in 1948, adding another cohort of 5,000 of their offspring and spouses in 1971, and several other cohorts of 5,000 since then. The risk models they computed are publicly available, and we describe some of them here. The study computed models for the 10-year risk of general cardiovascular disease (CVD), where the population of interest consists of individuals 30 to 74 years old and without CVD at the baseline examination. The 6 predictive variables (risk factors) are: age, diabetes, smoking, systolic blood pressure, cholesterol, and HDL cholesterol. The predictive model for women is given by the function

$$P_F(\{X_i\}) = 1 - 0.95012^{\exp(\sum_i \beta_i X_i - 26.1931)},$$

where exp() is the exponential function, β_i is the regression coefficient and X_i is the input for each risk factor. The risk for men is given as

$$P_M(\{X_i\}) = 1 - 0.88936^{\exp(\sum_i \beta_i X_i - 23.9802)}.$$

The regression coefficients for the model for women are given as follows:

$$\begin{aligned}\sum_i \beta_i X_i = 2.32888 \cdot \log(\mathrm{A}) + 1.20904 \cdot \log(\mathrm{C}) - 0.70833 \cdot \log(\mathrm{HDL}) \\ +2.76157 \cdot \log(\mathrm{SBP}) + 0.52873 \cdot \mathrm{S} + 0.69154 \cdot \mathrm{D},\end{aligned}$$

where A denotes the age, D denotes the presence of diabetes, S denotes whether or not an individual is a smoker, SBP denotes the systolic blood pressure, C denotes the cholesterol level, and HDL denotes the HDL cholesterol level. The above risk functions can be expressed as a Taylor series. Truncating the Taylor series gives us a polynomial with rational coefficients which can be used to approximate the risk.

3.2 Logistic Regression Model

A predictive equation to screen for diabetes was developed based on logistic regression in [13]. The equation was computed from data on more than $1,000$ Egyptian patients with no history of diabetes. The predictive variables used were: age (a), sex, BMI, number of hours since the last food or drink (PT: postprandial time), and random capillary plasma glucose level (RPG). The study was cross-validated on a sample of more than $1,000$ American patients. The predictive equation calculated is

$$P(x) = \frac{e^x}{e^x - 1},$$

with the following logistic regression parameters:

$$x = -10.0382 + 0.0331 \cdot \mathrm{a} + 0.0308 \cdot \mathrm{RPG} + 0.2500 \cdot \mathrm{PT} + 0.5620 \cdot (\text{if female}) + 0.0346 \cdot \mathrm{BMI},$$

where age is given in years, random plasma glucose (RPG) in mg/dl, and postprandial time (PT) in hours. Undiagnosed diabetes is predicted if the value is greater than 0.20 (20%). Thus only one digit of accuracy is required beyond the decimal point when computing the value of the predictive function approximately.

Logistic regression was also used to develop earlier Framingham Risk Functions [12], but has been superseded in [19] by models based on Cox proportional hazard analysis. To demonstrate logistic regression analysis, in our implementation we used the model computed in another study, which measured only 200 male patients, over an observation period which remains unclear [21]. The resulting logistic regression model predicted the likelihood to have a heart attack in an unspecified period, only for men. The six predictive variables are: age (a), height (ht), weight (wt), the systolic blood pressure (sys), the diastolic blood pressure (dia), and the cholesterol (chol) level. As above, the predictive model is given by the logistic regression function

$$P(x) = \frac{e^x}{e^x + 1},$$

where x is the sum of the variables weighted by the logistic regression coefficients, i.e.

$$x = 0.072 \cdot \mathrm{a} + 0.013 \cdot \mathrm{sys} - 0.029 \cdot \mathrm{dia} + 0.008 \cdot \mathrm{chol} - 0.053 \cdot \mathrm{ht} + 0.021 \cdot \mathrm{wt}.$$

This function can be approximated by a Taylor series

$$P(x) = \frac{e^x}{e^x + 1} = \frac{1}{2} + \frac{1}{4}x - \frac{1}{48}x^3 + \frac{1}{480}x^5 - \frac{17}{80640}x^7 + \frac{31}{1451520}x^9 + O(x^{11}).$$

Using terms in the Taylor expansion up to degree 7 we get roughly 2 digits of accuracy to the right of the decimal, which gives us an accurate percentage.

4 Practical Homomorphic Encryption

In 2009, Gentry [22] proposed the first fully homomorphic encryption (FHE) scheme. An FHE scheme makes it possible to encrypt data and then carry out arbitrary computations on the encrypted data by operating on ciphertexts only without the need to decrypt the data first, and without knowledge of the secret decryption key. The result of the computation is given in encrypted form and can only be decrypted by a legitimate owner of the private decryption key. The key to allowing arbitrary computations is that an FHE scheme allows both homomorphic addition and multiplication operations on the encrypted data. Previous homomorphic encryption schemes only provided one of the two operations and therefore were not fully homomorphic. Ciphertexts of current FHE schemes inherently contain a certain amount of noise, which grows during homomorphic operations. This noise "pollutes" the ciphertext and if it grows too large, makes correct decryption impossible, even with the legitimate decryption key. These schemes have at their core a so-called somewhat homomorphic encryption scheme that can handle a certain amount of homomorphic computation. To enable an unlimited number of operations, ciphertexts need to be refreshed by a costly recrypt procedure called bootstrapping.

In Gentry's initial work and many follow-up papers, the standard way of encrypting data is bitwise. This means that the encryption procedure takes each bit of the data separately and produces a corresponding ciphertext. Addition and multiplication of bits (modulo 2) corresponds to bitwise XOR and AND operations and thus allows to evaluate any boolean circuit, i.e. carry out arbitrary computation, by first expressing the computation in XOR and AND gates. But breaking down a computation into bit operations can quickly lead to a complicated and deep circuit that can not be handled by the somewhat homomorphic scheme and requires bootstrapping.

Functions such as those presented in the previous section, however, can be approximated by polynomial expressions in integer values, and do not necessarily need to be expressed in a bitwise manner. Some of the more efficient FHE schemes allow to encrypt polynomials that can encode such integer values. The advantage of this approach is that a single ciphertext now contains much more information than just a single bit of plaintext, but restricts the possible operations to arithmetic circuits in these polynomials. Furthermore, these functions are often simple enough such that the expensive bootstrapping procedure can be avoided.

The most efficient homomorphic encryption schemes of this type (see [23–26]) operate in polynomial rings and have their security based on hard problems in lattices, in particular the ring version of the learning with errors problem. These often provide a leveled homomorphic scheme, which can evaluate more initial computation than a somewhat homomorphic scheme before bootstrapping becomes necessary. In particular, when using a leveled homomorphic scheme it is possible, for a given computational task, to choose parameters that allow to evaluate the computation with the leveled scheme without bootstrapping. The solution we use in our implementation here is a lattice-based scheme from [26]. We describe how the scheme works in the following subsection.

4.1 The Homomorphic Encryption Scheme

In our implementation we use the more practical variant of the homomorphic encryption scheme recently proposed in [26, Section 5] (it is based on the schemes from [27, 28], with an improvement from [29]). The scheme is described in a specialized version below. It operates in the ring $R = \mathbb{Z}[X]/(X^n + 1)$ for n being a power of 2, i.e. the objects the scheme uses are all polynomials with integer coefficients of degree less than n. In particular, plaintexts and ciphertexts are such polynomials. An element $a \in R$ has the form $a = \sum_{i=0}^{n-1} a_i X^i$, $a_i \in \mathbb{Z}$. Computing on elements of R means that we add and multiply polynomials modulo $X^n + 1$. The key generation and encryption functions make use of two probability distributions χ_{key} and χ_{err} on R for generating small elements. The distribution χ_{key} is used in key generation, in this case it is the uniform distribution on polynomials with coefficients in $\{-1, 0, 1\}$. Sampling an element according to this distribution means sampling all its coefficients uniformly from $\{-1, 0, 1\}$. The distribution χ_{err} is a discrete Gaussian distribution, which is used to sample small noise or error polynomials (the main contributing terms to the noise inherent in ciphertexts mentioned above). For more specific details we refer to [26].The reader can simply think of elements sampled according to these distributions as polynomials in R with small integer coefficients.

The scheme is a public key encryption scheme and consists of the following algorithms: a key generation algorithm $\mathsf{KeyGen}(n, q, t, \chi_{\text{key}}, \chi_{\text{err}})$ that, on input the system parameters, generates a public/private key pair and an evaluation key, which is needed for the homomorphic multiplication operation and contains information about the private key in encrypted form; an encryption algorithm $\mathsf{Encrypt}(h, m)$ that encrypts a message m using the public key h; a decryption algorithm $\mathsf{Decrypt}(f, c)$ that decrypts a ciphertext c with the private key f and returns the plaintext message m that was encrypted in c; a homomorphic addition function $\mathsf{Add}(c_1, c_2)$ that produces a ciphertext encrypting the sum $m_1 + m_2 \in R/tR$ modulo t from two ciphertexts c_1 and c_2, encrypting the messages m_1 and m_2, respectively; and finally a homomorphic multiplication function that, given encryptions of m_1 and m_2, outputs a ciphertext encrypting the product $m_1 m_2$ modulo t. In more detail, these algorithms are given below. Note that some operations reduce coefficients of polynomials in R modulo an integer modulus q, this operation is denoted by $[\cdot]_q$. The plaintext space for encrypting is R/tR, i.e. polynomials in R with coefficients reduced modulo the integer plaintext modulus $t < q$. This means that data to be encrypted must be encoded as a polynomial $[m]_t$ (for $m \in R$) with coefficients reduced modulo t. The number $\lfloor q/t \rfloor$ is the largest integer not exceeding q/t.

- $\mathsf{KeyGen}(n, q, t, \chi_{\text{key}}, \chi_{\text{err}})$: On input the degree n and moduli q and t, the key generation algorithm samples small polynomials $f', g \leftarrow \chi_{\text{key}}$ from the key distribution and sets $f = [tf'+1]_q$. If f is not invertible modulo q, it chooses a new f'. It computes $h = [tgf^{-1}]_q$, where $f^{-1} \in R$ is the inverse of f modulo q, and outputs the public and private key pair $(\mathsf{pk}, \mathsf{sk}) = (h, f) \in R^2$ together with an evaluation key $\mathsf{evk} \in R^{\lceil \log_2(q) \rceil}$ that is used in the homomorphic multiplication algorithm. For more details see [26, Section 6].
- $\mathsf{Encrypt}(h, m)$: The encryption algorithm samples small error polynomials $s, e \leftarrow \chi_{\text{err}}$, and outputs the ciphertext $c = [\lfloor q/t \rfloor [m]_t + e + hs]_q \in R$.
- $\mathsf{Decrypt}(f, c)$: Given a ciphertext c and the private decryption key f, the decryption algorithm computes $m = \left[\left\lfloor \frac{t}{q} \cdot [fc]_q \right\rceil\right]_t \in R$.
- $\mathsf{Add}(c_1, c_2)$: Given two ciphertexts c_1 and c_2, output the ciphertext $c_{\text{add}} = [c_1 + c_2]_q$.
- $\mathsf{Mult}(c_1, c_2, \mathsf{evk})$: Given two ciphertexts c_1 and c_2 and the evaluation key evk, compute $\tilde{c}_{\text{mult}} = \left[\left\lfloor \frac{t}{q} c_1 c_2 \right\rceil\right]_q$ and output $\mathsf{KeySwitch}(\tilde{c}_{\text{mult}}, \mathsf{evk})$, where the key switching function $\mathsf{KeySwitch}$ is needed to transform the ciphertext so that it can be decrypted with the original secret key. Again, for more details, we refer to [26].

4.2 Encoding Real Numbers

For the applications described in Section, it is necessary to encode real numbers as elements of the encryption scheme's plaintext space. As mentioned above, recent papers [22, 29] presenting homomorphic encryption schemes usually assume that the data to be encoded and encrypted is handled bitwise, with a separate ciphertext for each bit (or sometimes with an optimization to pack many bits into each ciphertext as first described in [30] and used in [31]). But for example representing integers as a collection of bits requires a deep circuit just to do a simple integer multiplication when done bitwise. Before we describe how to encode real numbers, let us first consider integer values only. We use a technique for encoding integers which has already been described in [3] and has been used for machine learning algorithms in [9]. An integer is encoded as a polynomial $m \in R/tR$ (for the definition of R/tR see Section 4.1) via its bit representation. Namely, let $z \in Z$ be an integer and let $z = (\pm 1) \sum_{i=0}^{l} z_i 2^i$, $z_i \in \{0, 1\}$ be its binary representation, where $l = \lfloor \log_2(|z|) \rfloor + 1$. Then, as long as $l < n$ one can encode the plaintext message z as the polynomial $m = (\pm 1) \sum_{i=0}^{l} z_i X^i$. Such a polynomial can be decoded to give back an integer by evaluating it at 2, i.e. $z = m(2)$. For example, the integer 11 is encoded as the polynomial $1 + X + X^3$.

With this encoding, integer multiplication corresponds to direct multiplication of polynomials whenever t is large enough such that the coefficients of the resulting polynomial are all smaller than t and its degree does not exceed the maximal degree of polynomials in the ring used in the scheme. The latter two conditions need to be ensured, because homomorphic multiplication corresponds to multiplication of polynomials in R/tR. This is a polynomial multiplication followed by reductions modulo the ring polynomial $X^n + 1$ to reduce the degree below n and modulo t in the coefficients of the result. If these conditions are satisfied, a homomorphic multiplication of two ciphertexts corresponds directly to multiplication of the encrypted integers, not requiring a deep circuit to execute. Note that the internal representation of integers during and after a computation is redundant in that polynomials will represent integers via a 2-adic representation as above, but possibly with integer coefficients larger other than 0 or 1, depending on the specific computation. This does not change the fact that evaluation at 2 yields the corresponding integer result that is encoded by the polynomial.

For example, the polynomial $-3 + X + 3X^2$ also encodes the integer $-3 + 2 + 3 \cdot 2^2 = 11$ and could be the result of a computation on polynomials with binary coefficients.

Using the encoding technique for integers, we can now encode real numbers that are approximated to a given precision by rational numbers. For the entire computation, a fixed precision is chosen in advance, and real numbers at that precision are scaled by the corresponding factor, e.g. a power of 10 or a power of 2, to make them integers, which are then encoded as just described. At the end of the computation and after decryption, the result is converted back to a real number by dividing by that power of 10. How much precision is needed, depends on the accuracy required in the answer and on the complexity of the function to be computed. Also, the fact that we work with multiples of the actual data needs to be accounted for in the computation, which might have to be adjusted to produce the desired results.

The restriction to arithmetic circuits in R together with our encoding of numbers also means that we cannot carry out divisions of integers or real numbers. To avoid such divisions in our operations on encrypted data, we assume that our function is approximated by a polynomial with rational coefficients and that it is normalized by multiplying by an integer to clear all denominators. After decryption, the answer can be divided by the same integer to obtain the result.

4.3 How to Set Parameters

In practice, it is not necessary to use homomorphic encryption schemes that can handle arbitrary large amounts of computation (such as the fully-homomorphic encryption schemes mentioned earlier, which can be realized using the extremely costly bootstrapping step from Gentry's blueprint [22]). Instead, the required functionality can be achieved by choosing a leveled homomorphic scheme and fine-tuning the parameters to the specific computation, such that security and correctness are guaranteed. This leads to a more efficient scheme, but only allows for a finite amount of computation on encrypted data. The security requirement demands that the best known attacks to break the encryption scheme take at least a certain amount of time fixed in advance (also referred to as the security level). From that, bounds on the parameters are deduced to ensure this requirement. For a given desired amount of computation, the correctness condition is derived from the inherent noise growth, and also leads to bounds on the parameters, which have to be met to ensure that the inherent noise does not grow too large and decryption still works correctly. A combination of these two conditions leads to concrete intervals from which the parameters need to be chosen. The amount of computation allowed for a given parameter set, while still ensuring correctness and security, is fixed in advance, and depends on the size of the inputs and the complexity of the function to be computed, as well as on the homomorphic encryption scheme.

For the particular function considered here, we estimate the parameters based on the analysis given in the theorems and appendices of [26]. This approach is particularly well-suited for the application described here, direct-to-patient services, where there is a fixed function to be computed and a fixed amount of data input from the patient. Also, there is a cost to the above approach to encoding integers which we now explain. Our method of encoding allows for multiplication of integers but not division, thus the need to clear denominators to avoid fractions. We also need to ensure that our parameters are set large enough so that the computation on the encoded integers does not correspond to polynomials with coefficients exceeding the modulus t. If a reduction modulo t occurs in the plain text space, the computation does not represent the corresponding integer operations any more. Thus we set t to be a large power of 2, to allow for the coefficients of polynomials representing the results to grow that large. As t grows, to maintain security we must increase both the size of the modulus q, which determines the coefficient space for the polynomial ring, and the dimension of the lattice, which determines the degree of the polynomials to be handled. The performance of the scheme is proportional to the cost of multiplying polynomials in this polynomial ring, thus performance degrades as the complexity of the function and the size of the input grows. We follow the analysis in [26] and present two parameter sets in Table 1 that allow different amounts of computation. While the first parameter set $\mathbf{P_1}$ only allows a quadratic polynomial in the input data to be correctly evaluated, the second parameter set $\mathbf{P_2}$ can be used to evaluate polynomials of higher degree (such as 8).

	$\log_2(q)$	t	n
$\mathbf{P_1}$	128	2^{10}	2^{12}
$\mathbf{P_2}$	512	2^{40}	2^{14}

Table 1. We give two sets of parameters, $\mathbf{P_1}$ and $\mathbf{P_2}$, for choices of integer moduli q and t and the degree n in $R = \mathbb{Z}[X]/(X^n + 1)$. Parameter set $\mathbf{P_1}$ is chosen to allow the computation of the exponent in the logistic regression function. $\mathbf{P_2}$ is chosen to allow the computation of the Taylor series up to degree 7, to approximate the output of the logistic regression function. Both sets provide 80 bits of security, which means that the algorithms to break the scheme that are considered in [26] take time at least 2^{80}.

5 Performance Numbers

Due to the simple function we compute, and the small amount of data computed on, we have been able to implement a practical version of the homomorphic encryption scheme presented in [26]. As becomes clear from Table 1, even the small parameter set $\mathbf{P_1}$ requires computation with relatively large polynomials (in this case degree 4096) where each coefficient is of size 128 bits. When computing with such large polynomials it makes sense to switch from methods based on “schoolbook” multiplication (with run-time complexity $O(n^2)$ for degree n polynomials) to computation of the discrete Fourier transform using the fast Fourier transform (FFT) [32] (with a run-time complexity $O(n \log n)$ of arithmetic operations). Due to the special shape of the ring we use, $R = \mathbb{Z}[X]/(X^n + 1)$, where $n = 2^m$ is a power of 2, we can use the FFT algorithm by Nussbaumer [33] based on recursive negacyclic convolution (see [34, Exercise 4.6.4.59] for more details). The negacyclic convolution of two sequences $(x_0, x_1, \ldots, x_{2^m-1})$ and $(y_0, y_1, \ldots, y_{2^m-1})$ is the sequence $(z_0, z_1, \ldots, z_{2^m-1})$ with

$$z_k = x_0 y_k + \ldots + x_k y_0 - (x_{k+1} y_{2^m-1} + \ldots + x_{2^m-1} y_{k+1}).$$

Note that computing such a negacyclic convolution is equivalent to polynomial multiplication modulo $X^{2^m} + 1$ (where the different x_k are regarded as coefficients of a polynomial). The novelty of this algorithm is that it performs an FFT on the *coefficient polynomials*, see [35] for more details. Note that other asymptotic fast multiplication algorithms can be used as well, examples are the Schönhage-Strassen algorithm [36] or any other multiplication methods based on a number theoretic transform. We choose the algorithm from [33] since it is especially efficient for computations in $\mathbb{Z}[X]/(X^n + 1)$.

operation	$\mathbf{P_1}$	$\mathbf{P_2}$
encryption	13	577
decryption	11	549
addition of ciphertexts	< 1	1
multiplication of ciphertexts	39	5056

Table 2. Performance results for the core operations used in our homomorphic encryption using parameter set $\mathbf{P_1}$ and $\mathbf{P_2}$. The results are expressed in milliseconds. These results were obtained on a laptop (Intel Core i7-3520M at 2893.484 MHz).

We implemented and deployed the homomorphic scheme from [26] as a cloud service to compute on encrypted data, where we follow the steps as outlined in Figure 1 and we show a scenario which is typical of functions used for predictive analysis in health care. We have benchmarked the various core operations of this scheme on a modern laptop (Intel Core i7-3520M at 2893.484 MHz), the performance results are summarized in Table 2. All these core operations are stated in terms of milliseconds and can be considered practical. The encryption and decryption operations are performed by the user before and after sending the data to the cloud. The cloud receives the encrypted data, performs the additions and multiplications on this data in encrypted form, and sends the encrypted result back to the client. The size of the encrypted data is relatively large: using the parameters $\mathbf{P_1}$, a single encrypted value is 64 kilobytes (4096 coefficients of 128-bit). Thus the communication overhead with the cloud service provider (Microsoft's Windows Azure) dominates the time spent in the entire computation. In practice the timings for outsourcing the computation vary from less than a second up to two seconds: this is significantly more than the time spent on encrypting, decrypting and computing on the data, but still practical. Note that when using the parameter set $\mathbf{P_2}$, the size of one ciphertext grows to roughly one megabyte (16384 coefficients of 512-bit). In that case, the network overhead is more significant, but the computation cost is also greater since a multiplication of ciphertexts requires around five seconds with this scheme. Overall, computing the Taylor series up to degree 7 takes more than 30 seconds since computing the powers of x already requires 6 multiplications at 5 seconds each. Given that all other operations in the outsourced computation are rather cheap, the overall computation time for the cloud service including the network overhead is significantly below 1 minute.

Note that in our performance numbers based on the scheme in [26], the homomorphic multiplication operation includes a costly key switching step that increases the time for homomorphic multiplication, while decreasing the size of ciphertexts. It guarantees that ciphertexts do not grow and are decryptable by the original secret key. Thus there is a trade-off to consider when evaluating low-degree polynomials homomorphically, as in our current scenario. Namely, using a different scheme such as the one in [24] could result in significantly faster homomorphic multiplication, at the cost of allowing ciphertexts to grow during homomorphic multiplication. For a low-degree computation, it might not be beneficial to implement the modulus switching technique that is required for deeper circuits. Thus, overall, homomorphic multiplication might be significantly more efficient than the timings here suggest.

In a recent work [37] the homomorphic encryption scheme used here was also implemented. See [37] for performance details and the publicly available C++ software implementation.

6 Automatic Parameter Selection Module

The above discussion makes clear that an important aspect of designing such a system is the selection of appropriate parameters for implementing the homomorphic encryption scheme to ensure its security and correctness for the computation to be performed. We propose an automatic parameter selection module which computes these parameters. For the reasons explained in Section 4, we assume that *the function to be computed is approximated by a polynomial with rational coefficients.* And we assume also that denominators are cleared to transform the function into a polynomial with integer coefficients, that the amount of real precision that is required for the computation is fixed, and all real numbers are scaled to integers corresponding to that amount of precision. Then the module selects parameters depending on the following inputs:

1. the choice of a homomorphic encryption scheme,
2. the degree of the normalized polynomial to be evaluated and the size of its coefficients,
3. the size of the scaled inputs.

In this work, we use the more practical variant of the scheme from [26], but a similar analysis also applies to related schemes (e.g. the one in [25]).

In certain application scenarios for homomorphic encryption, the function to be computed might be proprietary. In that case, the parameters for the encryption scheme need to be provided by the function owner, along with specified bounds on the size of allowable inputs. We cannot accurately estimate the correct parameters without knowing the degree of the polynomia that is being evaluated and the size of its coefficients. Thus, for the remainder of our discussion and the detailed analysis of the function implemented in this paper, we assume that the function to be evaluated on encrypted data is public. This means that all its polynomial coefficients are known and each scalar multiplication can be implemented as an optimized sequence of additions of ciphertexts. The steps of parameter setting are as follows.

Correctness First, the *plaintext modulus* t must be chosen large enough such that it is bigger than the coefficients of the polynomial that encodes the integer result of the computation. This is important in order to guarantee that they are not reduced modulo t to something smaller. If integers are encoded as polynomials with binary coefficients (in the way shown in Section 4.2), then adding such a polynomial to itself s times results in a polynomial with integer coefficients at most s, where the degree of the polynomial remains the same. More general, when adding s polynomials with coefficients of size at most B, then the resulting polynomial has coefficients of size at most sB and its degree is the same. In contrast to the addition operation, multiplication of such polynomials results in much faster coefficient growth due to the multiplication of cross-terms. Note also that the degrees of the polynomials being multiplied add up to the degree of the product polynomial. So, how large do these coefficients get after L levels of multiplications, i.e. when evaluating a polynomial function (or a monomial) on encrypted data of degree 2^L? When starting with integers less than 2^d that are encoded as binary polynomials (of degree d), the coefficients of the resulting polynomial can be as large as $d^{2^L-1}2^{2^L-L-1}$, and its degree can be up to $2^L(d-1)$. Thus, to receive a meaningful result which reflects the actual compuation on integers, we need to ensure that $t > d^{2^L-1}2^{2^L-L-1}$, and also that the degree n is chosen large enough so as to be greater than $2^L(d-1)$. In fact, to evaluate a function represented by a polynomial of degree 2^L, we need to choose t somewhat larger than this bound to account for the scalar multiplications and additions.

As mentioned in Section 4, in most lattice-based encryption schemes, encryption involves adding *noise*, in the form of an error vector or error polynomial, to a (possibly noisy) inner product; thus the reason for the name of the hardness assumption learning with errors (LWE). As ciphertexts are added and multiplied, the noise term contained in the resulting ciphertexts grows. At the end of the computation, the overall noise must be small enough to still allow for correct decryption. This means that it must be small enough compared to the ratio of q to t, $\Delta = \lfloor q/t \rfloor$, in particular for the scheme we consider here, roughly less than $\Delta/2$ (see [26, Lemma 1]).

So for a given initial noise size and a given degree of the polynomial to be evaluated, the ratio of q to t has to be large enough to allow the amount of noise growth incurred by the computation. For given t and n, the size of the initial noise and a fixed number of levels of multiplications, using [26, Theorem 4 and Lemma 4] we can estimate the size of the resulting noise, and we can then set q large enough to allow for correct decryption ([26, Theorem 5]). We give more details below.

Security In addition to the above conditions to guarantee correctness, there is a delicate balance between these parameters q, t, n and the size of the noise to achieve the desired security. The best known attacks on lattice-based systems work by finding a short vector in a lattice. One can use such a short vector to distinguish pairs of uniform random ring elements from LWE samples. To distinguish an instance of the LWE problem from uniform random with advantage ϵ, an attacker needs to find a vector of length at most $\alpha(q/\sigma)$, where α is a constant which depends on ϵ and σ is the standard deviation of the error distribution. Thus for a given security requirement, the shortest vector obtained through the best lattice reduction algorithms must be longer than a vector which could give an adversary a non-negligible advantage in the Ring-LWE distinguishing problem. This gives us an inequality which must be satisfied involving q, n, and σ (see [26, Appendix K] for more details and the precise inequality, and [37, Section 3] for a more recent approach). In our example here (as in [26, Table 1]), we calculate parameters to achieve 80 bits of security, allowing a distinguishing advantage of 2^{-80}.

Taking into account all the bounds and requirements to achieve correctness and security for homomorphically evaluating a given function on inputs of a given size, we propose an automatic parameter selection module which determines the minimum parameters as required.

Algorithm to select parameters for a computation under homomorphic encryption

We outline an algorithm that works for the evaluation of polynomials in one variable. With slight alterations, the algorithm can be generalized to the case of multivariate polynomials. Let $F(z) = \sum_{i=0}^{\deg} c_i z^i$, $c_i \in \mathbb{Z}$ be the polynomial function to be evaluated under the homomorphic encryption scheme.

Input:

1. deg: the degree of $F(z)$, the polynomial to be evaluated on encrypted data, i.e. $\deg < 2^L$,
2. s: the maximum of the coefficients of $F(z)$, i.e. $c_i < s$, for all i,

3. d: the size of integers to be handled, i.e. all inputs are integers z with absolute value less than 2^d, encoded as polynomials with binary coefficients,
4. k: the desired security level, i.e. the best lattice attack runs in time $> 2^k$ and distinguishes with advantage less than 2^{-k},
5. σ: the standard deviation σ for the error distribution, typically $\sigma = 8$, determining the bound on the error distribution, B_{err}, typically 6σ,
6. B_{key}: the bound on the key distribution, which can be chosen to be 1 in practice,
7. w: the word size for the evaluation key, for example $w = 2^{32}$,
8. δ: the expansion factor, which is $\delta = n$ since n is a power of 2.

Output: parameters (t, q, n) for the homomorphic encryption scheme that ensure correctness and security.

Stage 1. In the first stage, set t large enough to allow for correctness of operations modulo t, i.e. such that no reductions modulo t occur in the coefficients of the plaintext polynomials. For multiplications only, the condition on t is $t > d^{2^L-1} \cdot 2^{2^L-L-1}$, as explained above. However, to evaluate the polynomial $F(z) = \sum_{i=0}^{\deg} c_i \cdot z^i$, we also need additions of ciphertexts. Here, the z^i correspond to polynomials with coefficients less than $d^{2^L-1} \cdot 2^{2^L-L-1}$. These z^i are multiplied by a scalar which is at most s and then these deg terms are added together, so the resulting coefficients are at most

$$s \cdot \deg \cdot d^{2^L-1} \cdot 2^{2^L-L-1} < s \cdot 2^L \cdot d^{2^L-1} \cdot 2^{2^L-L-1} = s \cdot d^{2^L-1} \cdot 2^{2^L-1}.$$

So set

$$t > s \cdot d^{2^L-1} \cdot 2^{2^L-1}.$$

Note that this assumes a naive way of multiplying by scalars. We also need to keep in mind that for multiplications to represent integer multiplications, we need the degree condition $n > 2^L \cdot (d-1)$ to avoid reductions in the ring modulo $X^n + 1$.

Stage 2. In this stage, we estimate the error growth to ensure correctness. Assume that t was chosen in Stage 1 and is now fixed. The final size e of the error must be less than $\Delta/2$, where $\Delta = \lfloor q/t \rfloor$, so set q such that $2e < \Delta$ which means that we roughly have $q > 2e \cdot t$. Theorem 4 and Lemma 4 in [26] show that multiplication of two ciphertexts with noise of size V results in a ciphertext with noise of size E, where

$$E < \delta t(4 + \delta t B_{key})V + \delta^2 t^2 B_{key}(B_{key} + t) + \delta^2 t \log_w(q) w B_{err} B_{key}.$$

In order to evaluate the function $F(z)$, which has degree $\deg < 2^L$, we carry out at most deg additions of terms of degree less than 2^L. Using the lower bound on Δ from [26, Theorem 5], and taking into account the noise growth resulting from the additions and coefficients of $F(z)$, we get the following lower bound on q to ensure correctness:

$$q > 2t \cdot s \cdot \deg \cdot (1+\epsilon_1)^{L-1} n^{2L} t^{2L-1} B_{key}^L \cdot \\ \left((1+\epsilon_1) n t^2 B_{\text{key}}(2B_{\text{err}} + t/2) + L(t(B_{key} + t) + \log_w(q) w B_{err})\right),$$

where $\epsilon_1 = 4(\delta t B_{key})^{-1}$, V has been replaced by the upper bound $\delta t B_{\text{key}}(2B_{\text{err}} + t/2)$ on the noise in a fresh ciphertext (see [26, Lemma 2]) and we have used $\delta = n$. Here, we see the dependence on $\delta = n$ in the noise bound that determines the bound on q. To satisfy security requirements, n must be much smaller than q. At this point, we can fix an initial value for n, for example $n = 2^{12}$, then set q according to the bound just given, and then proceed to the next stage to check that the bounds implied by security are satisfied.

Unfortunately, the above bound is not completely independent of q because of the term $\log_w(q)$. In practice, we choose w with respect to the expected size of q such that this term is rather small, so we can treat the bound as a lower bound on q. However, it should be checked with the concrete chosen parameters in the end.

Stage 3. In this stage, we assure that the desired security level is met. To distinguish with an advantage of ϵ in the RLWE problem, an adversary is required to find vectors of length at most $\alpha \cdot (q/\sigma)$ where $\alpha = \sqrt{\ln(1/\epsilon)/\pi}$. In our specific parameter examples, we use $\epsilon = 2^{-80}$, which results in $\alpha \approx 4.201$. We refer to [37] for a more complete description of a distinguishing attack and the precise lattices we are required to find short vectors in. Section 3 in [37] describes in detail a parameter selection approach that uses a simulation algorithm for the currently most efficient lattice-basis reduction algorithm BKZ 2.0 presented in [38]. To find a short vector in the n-dimensional target lattice, it is embedded into a lattice of higher dimension $m > n$. For given m and a fixed security level λ (e.g. $\lambda = 80$), this method determines the minimal root-Hermite factor $\gamma(m)$ that can be achieved running BKZ 2.0 for time 2^λ. The factor $\gamma(m)$ determines the size of the shortest vector that can be found with that specific setting of the BKZ 2.0 algorithm, which is $\gamma(m)^m \cdot q^{n/m}$. To distinguish with advantage ϵ, one needs $\gamma(m)^m \cdot q^{n/m} = \alpha \cdot (q/\sigma)$. This means that for security we require (see [37, Section 3.3])

$$\log_2(q) \leq \min_{m>n} \frac{m^2 \cdot \log_2(\gamma(m)) + m \cdot \log_2(\sigma/\alpha)}{m - n},$$

and in terms of q, this yields

$$q \leq q_{\max}(n, \epsilon) := \min_{m>n} 2^{\frac{m^2 \cdot \log_2(\gamma(m)) + m \cdot \log_2(\sigma/\alpha)}{m-n}}.$$

In order to determine $q_{\max}(n, \epsilon)$, various $\gamma(m)$ values for different security levels and dimensions can be found in [37, Table 1].

Stage 4. In this final stage, we combine the bounds from the previous two stages to choose n and q. The computations in Stages 2 and 3 yield an upper and lower bound on q, both of which involve n. To ensure a simultaneous solution, we set n to a power of two such that

$$\begin{aligned} q_{\max}(n, \epsilon) > {} & 2t \cdot s \cdot \deg \cdot (1 + \epsilon_1)^{L-1} \delta^{2L} t^{2L-1} B_{key}^L \cdot \\ & \left((1 + \epsilon_1)\delta t^2 B_{\text{key}}(2B_{\text{err}} + t/2) + L(t(B_{key} + t) + \log_w(q) w B_{err})\right), \end{aligned}$$

i.e. such that we get a non-trivial interval for q. Finally, we select an appropriate q in this interval.

Example Using the model for predicting the likelihood of having a heart attack given in [21], and normalizing the functions and inputs as described above, we evaluate the function

$$F(z) = -17z^7 + 168z^5 - 1680z^3 + 20160z + 40320,$$

where the input z is computed as

$$z = 72 \cdot \mathrm{a} + 13 \cdot \mathrm{sys} - 29 \cdot \mathrm{dia} + 8 \cdot \mathrm{chol} - 53 \cdot \mathrm{ht} + 21 \cdot \mathrm{wt}.$$

Inputs to z are encoded as polynomials as described above.

The input to F is the sum of 6 scaled values, where the scalars are all less than 72 and the inputs are at most 400. So each input can be represented as a polynomial of degree at most 8 with binary coefficients, and each scalar multiplication can be achieved through at most 72 additions. So the result is represented by a ciphertext corresponding to a polynomial of degree at most 8 with coefficients at most $6 \cdot 72$, and taking into account the error growth for homomorphic additions, the computation of the input to the function F can be achieved with a small value of t, for example $t = 2^{10}$ will suffice.

The input to the function F is at most $6 \cdot 72 \cdot 400$ (which is an overestimate), and could thus be represented as a polynomial of degree at most $d = \lfloor \log_2(6 \cdot 72 \cdot 400) \rfloor = 17$ with binary coefficients. But since this input is not a freshly encrypted ciphertext, i.e. is not a freshly encoded integer, its coefficients may already be larger. Namely, the output of the homomorphic additions in the first step result in a ciphertext corresponding to a polynomial of degree 8 with coefficients up to size 432. Also, this corresponds to an integer which is 1000 times what the actual input is supposed to be, so the coefficients of F need to be scaled to account for this, making them even larger once they are normalized. One alternative is to have the client reencrypt the input to the function F. If the client inputs a freshly encrypted value for x where the ciphertext is represented as a polynomial of degree at most $d = 17$ with binomial coefficients, then to evaluate the function F on this input requires 3 levels of multiplication, so using the lower bound on t for $L = 3$ given above, $t > 8d^7$, we see that to achieve correctness for this computation we need to set t to be at least $t = 2^{32}$.

7 Conclusion

In this paper, we demonstrated a working implementation of a cloud service for performing private predictive analysis tasks on encrypted health data using homomorphic encryption. The cloud service makes predictions while handling only encrypted data, learning nothing about the submitted confidential medical data. In addition to the efficient implementation of the homomorphic encryption scheme and the cloud service, the main contribution of this work is to give details of using a practical homomorphic encryption scheme for real-life predictive analysis. Namely, we propose an automatic parameter selection module for determining safe parameters for the practical homomorphic encryption scheme to assure correctness and security of the results when evaluating functions used in predictive analysis such as logistic regression and Cox proportional hazard regression. We also indicate the extent to which such functions are prevalent in predictive analytics in health care, and provide an overview of scenarios for private computation where such functions are relevant. Future work will focus on improving the performance and increasing the scalability of systems to operate on encrypted health data. This includes improving the performance of practical homomorphic encryption schemes at scale and enlarging the class of functions which practical homomorphic encryption schemes can successfully evaluate.

Software Demonstration A live demo of the prediction service is available at the link given in [39].

Acknowledgements. We thank Craig Costello, Robert Dunnill, and Seny Kamara for help in preparing the demo, Adriana Lopez-Alt for comments on an earlier draft, and the anonymous referees for helpful suggestions to improve the exposition.

References

1. J. Benaloh, M. Chase, E. Horvitz, K. Lauter, Patient controlled encryption: ensuring privacy of electronic medical records, in: Proceedings of the first ACM Cloud Computing Security Workshop – CCSW, ACM, 2009, pp. 103–114.
2. S. Narayan, M. Gagné, R. Safavi-Naini, Privacy preserving EHR system using attribute-based infrastructure, in: Proceedings of the 2nd ACM Cloud Computing Security Workshop – CCSW, ACM, 2010, pp. 47–52.
3. K. Lauter, M. Naehrig, V. Vaikuntanathan, Can homomorphic encryption be practical?, in: Proceedings of the 3rd ACM Cloud Computing Security Workshop – CCSW, ACM, 2011, pp. 113–124.
4. M. Gymrek, A. L. McGuire, D. Golan, E. Halperin, Y. Erlich, Identifying personal genomes by surname inference, Science 339 (6117) (2013) 321–324.
5. T. Sumner, How to hide your genome, ScienceNow, `http://news.sciencemag.org/biology/2014/02/how-hide-your-genome` (2014).
6. C. Dwork, A firm foundation for private data analysis, Commun. ACM 54 (1) (2011) 86–95.
7. C. Clifton, T. Tassa, On syntactic anonymity and differential privacy, in: C. Y. Chan, J. Lu, K. Nørvåg, E. Tanin (Eds.), ICDE Workshops, IEEE Computer Society, 2013, pp. 88–93.
8. G. E. Simon, J. Unützer, B. E. Young, H. A. Pincus, Large medical databases, population-based research, and patient confidentiality, American Journal of Psychiatry 157 (11) (2000) 1731–1737.
9. T. Graepel, K. Lauter, M. Naehrig, ML confidential: Machine learning on encrypted data, in: T. Kwon, M.-K. Lee, D. Kwon (Eds.), Information Security and Cryptology – ICISC, Vol. 7839 of Lecture Notes in Computer Science, Springer, 2012, pp. 1–21.
10. R. Gennaro, C. Gentry, B. Parno, Non-interactive verifiable computing: Outsourcing computation to untrusted workers, in: T. Rabin (Ed.), CRYPTO, Vol. 6223 of Lecture Notes in Computer Science, Springer, 2010, pp. 465–482.
11. B. Parno, J. Howell, C. Gentry, M. Raykova, Pinocchio: Nearly practical verifiable computation, in: IEEE Symposium on Security and Privacy, IEEE Computer Society, 2013, pp. 238–252.
12. R. B. D'Agostino, M. J. Pencina, J. M. Massaro, S. Coady, Cardiovascular disease risk assessment: Insights from Framingham, Global heart 8 (1) (2013) 11–23.
13. B. P. Tabaei, W. H. Herman, A multivariate logistic regression equation to screen for diabetes development and validation, Diabetes Care 25 (11) (2002) 1999–2003.
14. C. R. Boyd, M.-A. Tolson, W. S. Copes, Evaluating trauma care: the TRISS method, The Journal of Trauma and Acute Care Surgery 27 (4) (1987) 370–378.
15. M. P. LaValley, Logistic regression, Circulation 117 (18) (2008) 2395–2399.
16. R. Blankstein, R. P. Ward, M. Arnsdorf, B. Jones, Y.-B. Lou, M. Pine, Female gender is an independent predictor of operative mortality after coronary artery bypass graft surgery contemporary analysis of 31 midwestern hospitals, Circulation 112 (9 suppl) (2005) 323–327.
17. S. Boekholdt, F. M. Sacks, J. W. Jukema, J. Shepherd, D. J. Freeman, A. D. McMahon, F. Cambien, V. Nicaud, G. J. De Grooth, P. J. Talmud and others, Cholesteryl ester transfer protein TaqIB variant, high-density lipoprotein cholesterol levels, cardiovascular risk, and efficacy of pravastatin treatment, Circulation 111 (3) (2005) 278–287.
18. A. Festa, K. Williams, A. J. Hanley, J. D. Otvos, D. C. Goff, L. E. Wagenknecht, S. M. Haffner, Nuclear magnetic resonance lipoprotein abnormalities in prediabetic subjects in the insulin resistance atherosclerosis study, Circulation 111 (25) (2005) 3465–3472.
19. R. B. D'Agostino, R. S. Vasan, M. J. Pencina, P. A. Wolf, M. Cobain, J. M. Massaro, W. B. Kannel, Circulation (6) 743–753.
20. D. J. Llewellyn, I. A. Lang, J. Xie, F. A. Huppert, D. Melzer, K. M. Langa, Framingham stroke risk profile and poor cognitive function: a population-based study, BMC neurology 8 (1) (2008) 12.
21. C. Flowers, RSCH 8140–Multivariate Statistics (Doctoral), See `http://coedpages.uncc.edu/cpflower/rsch8140/logistic_regression_example.htm` (2013).
22. C. Gentry, Fully homomorphic encryption using ideal lattices, in: STOC, 2009, pp. 169–178.
23. Z. Brakerski, V. Vaikuntanathan, Fully homomorphic encryption from ring-LWE and security for key dependent messages, in: P. Rogaway (Ed.), Advances in Cryptology – CRYPTO, Vol. 6841 of Lecture Notes in Computer Science, 2011, pp. 505–524.
24. Z. Brakerski, C. Gentry, V. Vaikuntanathan, Fully homomorphic encryption without bootstrapping, Electronic Colloquium on Computational Complexity (ECCC) 18 (2011) 111.
25. J. Fan, F. Vercauteren, Somewhat practical fully homomorphic encryption, Cryptology ePrint Archive, Report 2012/144, `http://eprint.iacr.org/` (2012).

26. J. W. Bos, K. Lauter, J. Loftus, M. Naehrig, Improved security for a ring-based fully homomorphic encryption scheme, in: M. Stam (Ed.), Cryptography and Coding - 14th IMA International Conference, IMACC 2013, Vol. 8308 of Lecture Notes in Computer Science, Springer, 2013, pp. 45–64.
27. D. Stehle, R. Steinfeld, Making NTRU as secure as worst-case problems over ideal lattices, in: Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vol. 6632 of Lecture Notes in Computer Science, Springer, 2011, p. 27.
28. A. López-Alt, E. Tromer, V. Vaikuntanathan, On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption, in: STOC, 2012, pp. 1219–1234.
29. Z. Brakerski, Fully homomorphic encryption without modulus switching from classical GapSVP, in: Advances in Cryptology - Crypto 2012, Vol. 7417 of Lecture Notes in Computer Science, Springer, 2012, pp. 868–886.
30. N. Smart, F. Vercauteren, Fully homomorphic SIMD operations, Designs, Codes and Cryptography (2012) 1–25doi:10.1007/s10623-012-9720-4.
URL `http://dx.doi.org/10.1007/s10623-012-9720-4`
31. C. Gentry, S. Halevi, N. P. Smart, Homomorphic evaluation of the AES circuit, in: Advances in Cryptology - Crypto 2012, Vol. 7417 of Lecture Notes in Computer Science, Springer, 2012, pp. 850–867.
32. J. W. Cooley, J. W. Tukey, An algorithm for the machine calculation of complex Fourier series, Mathematics of Computation 19 (1965) 297–301.
33. H. J. Nussbaumer, Fast polynomial transform algorithms for digital convolution, Acoustics, Speech and Signal Processing, IEEE Transactions on 28 (2) (1980) 205–215.
34. D. E. Knuth, Seminumerical Algorithms, 3rd Edition, The Art of Computer Programming, Addison-Wesley, Reading, Massachusetts, USA, 1997.
35. H. J. Nussbaumer, Fast Fourier Transform and Convolution Algorithms, Springer, New York, 1982.
36. A. Schönhage, V. Strassen, Schnelle multiplikation großer zahlen, Computing 7 (3-4) (1971) 281–292.
37. T. Lepoint, M. Naehrig, A comparison of the homomorphic encryption schemes FV and YASHE, Cryptology ePrint Archive, Report 2014/062, `http://eprint.iacr.org/` (2014).
38. Y. Chen, P. Q. Nguyen, BKZ 2.0: Better lattice security estimates, in: ASIACRYPT, Vol. 7073 of Lecture Notes in Computer Science, Springer, 2011, pp. 1–20.
39. Y. Erlich, K. Lauter, J. Wilbanks, Encrypting genetic data for smarter sharing, AAAS 2014 Annual Meeting Press Briefings, `http://new.livestream.com/AAASmtg/Sunday`, (minutes 12:52–16:20) (2014).