Can Homomorphic Encryption be Practical?

Kristin Lauter[1], Michael Naehrig[1,2], and Vinod Vaikuntanathan[1,3*]

[1] Microsoft Research
klauter@microsoft.com
[2] Eindhoven University of Technology
michael@cryptojedi.org
[3] University of Toronto
vinodv@cs.toronto.edu

Abstract. The prospect of outsourcing an increasing amount of data storage and management to cloud services raises many new privacy concerns for individuals and businesses alike. The privacy concerns can be satisfactorily addressed if users encrypt the data they send to the cloud. If the encryption scheme is homomorphic, the cloud can still perform meaningful computations on the data, even though it is encrypted.

In fact, we now know a number of constructions of *fully* homomorphic encryption schemes that allow arbitrary computation on encrypted data. In the last two years, solutions for fully homomorphic encryption have been proposed and improved upon, but it is hard to ignore the elephant in the room, namely efficiency – *can homomorphic encryption ever be efficient enough to be practical?* Certainly, it seems that all known fully homomorphic encryption schemes have a long way to go before they can be used in practice. Given this state of affairs, our contribution is two-fold.

First, we exhibit a number of real-world applications, in the medical, financial, and the advertising domains, which require only that the encryption scheme is "somewhat" homomorphic. Somewhat homomorphic encryption schemes, which support a limited number of homomorphic operations, can be much faster, and more compact than fully homomorphic encryption schemes.

Secondly, we show a *proof-of-concept* implementation of the recent somewhat homomorphic encryption scheme of Brakerski and Vaikuntanathan, whose security relies on the "ring learning with errors" (Ring LWE) problem. The system is very efficient, and has reasonably short ciphertexts. Our *unoptimized* implementation in MAGMA enjoys comparable efficiency to even *optimized* pairing-based schemes with the same level of security and homomorphic capacity. We also show a number of *application-specific* optimizations to the encryption scheme, most notably the ability to convert between different message encodings in a ciphertext.

Keywords: Fully Homomorphic Encryption, Ring Learning with Errors, Lattices, MAGMA

1 Introduction

The development of cloud storage and computing platforms allows users to outsource storage and computations on their data, and allows businesses to offload the task of maintaining data-centers. However, concerns over loss of privacy and business value of private data is an overwhelming barrier to the adoption of cloud services by consumers and businesses alike. An excellent way to assuage these privacy concerns is to store all data in the cloud encrypted, and perform computations on encrypted data. To this end, we need an encryption scheme that allows meaningful computation on encrypted data, namely a *homomorphic encryption scheme.*

[*] This research was conducted while the second and third authors were at Microsoft Research Redmond. Full version of the ACM CCSW 2011 paper.

Homomorphic encryption schemes that allow simple computations on encrypted data have been known for a long time. For example, the encryption systems of Goldwasser and Micali [GM82], El Gamal [El-84], Cohen and Fischer [CF85], and Paillier [Pai99] support either adding or multiplying encrypted ciphertexts, *but not both operations at the same time.* Boneh, Goh and Nissim [BGN05] were the first to construct a scheme capable of performing both operations at the same time – their scheme handles an arbitrary number of additions but just *one* multiplication. More recently, in a breakthrough work, Gentry [Gen09,Gen10] constructed a *fully homomorphic encryption scheme* (FHE) capable of evaluating an arbitrary number of additions and multiplications (and thus, compute any function) on encrypted data.

The main point of this paper is to show to what extent current schemes can actually be used to compute functions of practical interest on encrypted data. Since the appearance of Gentry's scheme, there has been much informal discussion in the industry as to whether fully homomorphic encryption is implementable and practical. While the initial solution may not have been practical, subsequent developments produced other schemes [DGHV10,SV10,SS10] leading up to the most recent solutions of Brakerski and Vaikuntanathan [BV11b,BV11a], an implementation of which we consider in this paper. The scheme is efficient and simple, produces short ciphertexts, and its security is based on the ring learning with errors (Ring LWE) problem [LPR10].

While the performance of the state-of-the art FHE implementations is itself a question of interest (and has indeed been considered recently in, e.g., [GH11,SV11]), our focus here is on describing concrete practical applications and concrete useful functions to be computed, most of which require only a limited number of multiplications of ciphertexts (as well as a possibly very large number of additions of ciphertexts). For these applications, it is enough to consider an implementation of a somewhat homomorphic encryption (SwHE) scheme, namely, one which allows a fixed number of multiplications of ciphertexts. SwHE schemes are the building blocks for the FHE schemes of, e.g., [Gen09,DGHV10,BV11b,BV11a], and provide much better efficiency guarantees than their fully homomorphic counterparts.

1.1 Practical Applications for Homomorphic Encryption

We describe a number of concrete applications and functions to be implemented to provide cloud services in the medical, financial, and advertising sectors. (We provide a sketch of the applications here, and refer the reader to Section 2 for detailed descriptions.)

For a cloud service managing electronic medical records (EMR), consider a futuristic scenario where devices continuously collect vital health information, and stream them to a server who then computes some statistics (over these measurements, and over the course of time) and presumably decides on the course of treatment (e.g., whether the dosage of medicine should be changed). The volume of the data involved is large, and thus, the patient presumably does not want to store and manage all this data locally; she may prefer to use cloud storage and computation. To protect patient privacy, all the data is uploaded in encrypted form, and thus the cloud must perform operations on the encrypted data in order to return (encrypted) alerts, predictions, or summaries of the results to the patient.

We describe scenarios such as the above, which require computing simple statistical functions such as the mean, standard deviation, as well as logistical regressions that are typically used for prediction of likelihoods of certain desirable or undesirable outcomes. For these functions, it suffices to have a somewhat homomorphic encryption system which computes many additions and a *small number of multiplications* on ciphertexts: for example, averages require no multiplications, standard deviation requires one multiplication, and predictive analysis such as logistical regression requires a few multiplications (depending on the precision required). Other applications we describe in the financial and advertising sector use similar functions, except that in those sectors, the function itself may also be private or proprietary.

1.2 Our Implementation

We implemented the somewhat homomorphic encryption scheme from [BV11b] by Brakerski and Vaikuntanathan using the computer algebra system Magma [BCP97]. We ran experiments on an *ordinary laptop* with an Intel Core 2 Duo processor running at 2.1 GHz, with 3MB L2 cache and 1GB of memory, using Magma's polynomial arithmetic for all computations in the ring of polynomials modulo a prime q (denoted R_q).

The first problem we face is that of choosing secure parameters for the encryption scheme. Unlike traditional number-theory based systems, cryptosystems based on lattices and the learning with errors (LWE) problem tend to be governed by a number of inter-related parameters. We follow the methodology of Lindner and Peikert [LP11] (following earlier work of [GN08,MR09,RS10]) to choose these parameters correctly and securely.[4] More precisely, fixing a parameter D – an upper bound on the number of multiplication operations that the scheme supports – we compute parameters of the scheme secure against an attacker with a time-to-advantage ratio of about 2^{120} (or larger) which, by the heuristics of Lenstra and Verheul [LV01], roughly translates to a security level matching that of AES-128.

For parameter settings which support a single multiplication (i.e., $D = 2$) followed by a large number of (integer) additions, the underlying ring in our scheme is $R_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$ where $n = 2048$ and $q \approx 2^{58}$ is prime. We always set the error distribution for ring LWE to be a discrete Gaussian with standard deviation $\sigma = 8$. In fact, for these choices of parameters, we get more security than we asked for, namely a time-to-advantage ratio of about 2^{196}. [5] For this setting, the key and ciphertext sizes, and the running times are as follows:

- The public key is $2n \lg q \approx 29$ KB, the secret key is $n \lg q \approx 14$ KB, and a ciphertext generated by the encryption algorithm is $2n \lg q \approx 29$ KB. The ciphertext has two ring elements in $R_q = \mathbb{Z}_q[x]/\langle f(x)\rangle$, however homomorphic multiplication adds another ring element to the ciphertext making it $3n \log q \approx 43.5$ KB.
- Key-generation runs in 250 milliseconds (ms), encryption takes 24 ms, whereas decryption takes 15–26 ms (depending on whether we are decrypting a 2- or 3-element ciphertext). Homomorphic addition is essentially instantaneous (i.e., takes less than 1 ms), whereas homomorphic multiplication takes about 41 ms.

We note that polynomial multiplication modulo $x^n + 1$ (i.e. multiplication in R_q) takes 11 ms whereas polynomial addition is very cheap with less than 1 ms. The encryption function needs to do 2 multiplications in R_q, which are responsible for the major part of the computation time for encryption. Any improvement in polynomial modular multiplication thus will have drastic effects on the efficiency of the scheme.

We also implement an optimization proposed in [BV11a] – termed *relinearization* – that reduces the size of the ciphertext to two ring elements. This comes at a cost of increasing the keys and the time for homomorphic multiplication, although we observe that the advantage of this optimization outweighs the cost in the larger parameter settings.

Let us contrast these numbers with the performance of the BGN encryption scheme [BGN05] based on bilinear pairings on elliptic curves, which also supports one multiplication. While the key sizes in our scheme are larger, our homomorphic multiplication is much faster than that for the BGN scheme, which requires a pairing computation. A MAGMA implementation of the optimal ate pairing on a 254-bit BN (Barreto-Naehrig) curve takes around 240 ms on the same machine that we used to time the somewhat homomorphic scheme. The BGN scheme requires composite order pairing groups and thus, it is likely that the parameters are less favorable, leading to an even slower pairing computation. Even if we assume highly optimized pairings at the 128-bit security level (about 1ms per pairing) and the use of a prime-order version of the BGN scheme [Fre10] which requires at least 4 such pairings, the performance of the homomorphic multiplication

[4] Lindner and Peikert, and indeed all previous work, analyze security for the LWE problem, whereas here, we rely on the security of the more restricted Ring LWE problem. We still choose to use the analysis of [LP11] since (as pointed out therein), we do not know any attacks against Ring LWE that perform better than the attacks on LWE.

[5] Since we constrain the dimension n to be a power of 2, we sometimes get "jumps" in security levels. In such a case, we choose to use the smallest setting whose security level exceeds our minimum, namely a time-to-advantage ratio of 2^{120}.

of our MAGMA implementation is comparable to the BGN scheme. We remark that a C implementation (possibly using special purpose polynomial arithmetic along the lines of the SWIFFT hash function [LMPR08]) would likely be several orders of magnitude faster than the MAGMA implementation that we report here.

We build on the somewhat homomorphic encryption, and implement simple statistics such as mean, standard deviation and logistical regression, and report on the performance numbers (See Section 5 for more details). To support computing these functions, we present two tricks to encode messages appropriately. As such, the encryption scheme natively supports addition and multplication modulo a fixed integer t, where t is a parameter of the system. The first trick shows how to encode integers in a ciphertext so as to enable efficient computation of their sums and products *over the integers*. This is useful in computing the mean, the standard deviation and other private statistics efficiently. The second trick shows how to "pack" n encryptions of bits into a single encryption of the n-bit string. Some homomorphic operations, e.g., comparison of integers or private information retrieval, seem to require bit-wise encryptions of the input. Once the answers are computed, though, they can be packed into a single encryption using this trick.

Finally, the work of [BV11a] shows how to make this scheme *fully homomorphic* under the same assumption (Ring LWE) in an efficient way. Implementing the "bootstrapping" operation for this system (thus, making it fully homomorphic) is a very interesting avenue for future implementation efforts.

2 Cloud Services

Adoption of cloud services by consumers and businesses is limited by concerns over the loss of privacy or business value of their private data. In this section we will describe concrete and valuable applications of Somewhat Homomorphic Encryption which can help preserve customer privacy while outsourcing various kinds of computation to the cloud. In all of these scenarios, we imagine a future of streaming data from multiple sources, uploaded in encrypted form to the cloud, and processed by the cloud to provide valuable services to the content owner. There are two aspects of the computation to consider: the data itself, and the function to be computed on this data. We consider cases where one or both of these are private or proprietary and should not be shared with the cloud.

In all of these applications, we consider a single content-owner, who is the consumer for the cloud service. All data that is encrypted and sent to the cloud is public-key encrypted to the content-owner's public key, using the semantically secure somewhat homomorphic encryption scheme from [BV11b] described later in this paper.

2.1 Medical Applications: Private data and Public functions

In [BCHL09], a private cloud medical records storage system (Patient Controlled Encryption) was proposed, in which all data for a patient's medical record is encrypted by the healthcare providers before being uploaded to the patient's record in the cloud storage system. The patient controls sharing and access to the record by sharing secret keys with specific providers (features include a hierarchical structure of the record, ability to search the encrypted data, and various choices for how to handle key distribution). However this system does not provide for the cloud to do any computation other than search (exact keyword match, or possibly conjunctive searches). With our FHE implementation, we add the ability for the cloud to do computation on the encrypted data on behalf of the patient. Imagine a future where monitors or other devices may be constantly streaming data on behalf of the patient to the cloud. With FHE, the cloud can compute functions on the encrypted data and send the patient updates, alerts, or recommendations based on the received data.

The functions to be computed in this scenario may include averages, standard deviations or other statistical functions such as logistical regression which can help predict the likelihood of certain dangerous health episodes. Encrypted input to the functions could include blood pressure or heart monitor or blood sugar readings, for example, along with information about the patient such as age, weight, gender, and other risk factors. The functions computed may not need to be private in this case since they may be a matter of public health and thus public.

2.2 Financial Applications: Private data and Private functions

In the financial industry there is a potential application scenario in which both the data and the function to be computed on the data is private and proprietary.

As an example, data about corporations, their stock price or their performance or inventory is often relevant to making investment decisions. Data may even be streamed on a continuous basis reflecting the most up-to-date information necessary for making decisions for trading purposes. Functions which do computations on this data may be proprietary, based on new predictive models for stock price performance and these models may be the product of costly research done by financial analysts, so a company may want to keep these models private to preserve their advantage and their investment.

With FHE, some functions can be evaluated privately as follows. The customer uploads an encrypted version of the function to the cloud, for example a program where some of the evaluations involve encrypted inputs which are specified. The streaming data is encrypted to the customer's public key and uploaded to the cloud. The cloud service evaluates the private function by applying the encrypted description of the program to the encrypted inputs it receives. After processing, the cloud returns the encrypted output to the customer.

2.3 Advertising and Pricing

Imagine an advertiser, for example a cosmetics company, who wants to use contextual information to target advertising to potential customers. The consumer uses a mobile phone as a computing device, and the device constantly uploads contextual information about the consumer, including location, the time of day, information from email or browsing activity such as keywords from email or browser searches. In the future, imagine that information is uploaded potentially constantly from video devices: either pictures of objects of interest such as brands or faces which are automatically identified, or from a video stream from a camera on the body which is identifying context in the room (objects, people, workplace vs. home vs. store). When contextual information is uploaded to the cloud server and made accessible to the cosmetics company, the company computes some function of the contextual data and determine which targeted advertisement to send back to the consumer's phone.

Some examples of where context is important for advertising or providing targeted coupons: beer commercials during sports events, or, you are near a Starbucks in the morning and a coffee discount coupon for the Starbucks nearby is sent to your phone, or, cosmetics companies market different products for different times of day (e.g. Friday night going out vs. Sunday morning hanging out with the family), ads or coupons for shows if you are in New York near Broadway in the evening. Other (private) contextual data might be: your income, your profession, your purchasing history, your travel history, your address, etc.

Encrypted version: The problem with these scenarios is the invasion of privacy resulting from giving that much detailed information about the consumer to the server or to the advertising company. Now, imagine an encrypted version of this entire picture. All the contextual data is encrypted and then uploaded to the server; the advertiser uploads encrypted ads to the server; the server computes a function on the encrypted inputs which determines which encrypted ad to send to the consumer; this function could be either private/proprietary or not. All contextual data and all ads are encrypted to the consumer's public key. Then the cloud can operate and compute on this data, and the consumer can decrypt the received ad. As long as the cloud service provider does not collude with the advertisers, and semantically secure FHE encryption is employed, the cloud and the advertisers don't learn anything about the consumer's data. [6]

2.4 Functions to be computed with FHE

We can compute the following functions with a somewhat homomorphic encryption scheme:

- Average of n terms $\{c_i\}$: as a pair $(\sum_{i=1,\dots n} c_i, n)$, where $m = \frac{\sum_{i=1,\dots n} c_i}{n}$ is the average.
- Standard deviation: $\sqrt{\frac{\sum_{i=1,\dots n}(c_i-m)^2}{n}}$, returned as a pair which is the numerator and denominator of the expression, before taking the square root.
- Logistical regression: $x = \sum_{i=1,\dots n} \alpha_i x_i$, where α_i is the weighting constant or regression coeffficient for the variable x_i, and the prediction is $f(x) = \frac{e^x}{1+e^x}$

A couple of remarks are in order. First, we set the parameter choices for the encryption system based on the expected number of multiplication operations to be done to compute the given functions. These parameter choices determine the efficiency and security of the system. Thus parameters for the system need to be changed as the functions to be computed change.

Secondly, so far we do not have a way to efficiently do divisions of real numbers or square roots. Thus in the above computations, numerators and denominators need to be returned as separate encryptions.

3 The Encryption Scheme

We describe the ring learning with errors (Ring LWE) assumption of [LPR10] in Section 3.1, and present the somewhat homomorphic encryption scheme of [BV11b] based on Ring LWE in Section 3.2. We report on an instantiation of the parameters, as well as the running-times and sizes of the keys and ciphertexts in Section 5.

3.1 The Ring LWE Assumption

In this section, we describe a variant of the ring learning with errors (RLWE) assumption of Lyubashevsky, Peikert and Regev [LPR10]. In the RLWE assumption, we consider rings $R := \mathbb{Z}[x]/\langle f(x)\rangle$ and $R_q := R/qR$ for some degree n integer polynomial $f(x) \in \mathbb{Z}[x]$ and a prime integer $q \in \mathbb{Z}$. Note that $R_q \equiv \mathbb{Z}_q[x]/\langle f(x)\rangle$, i.e. the ring of degree n polynomials modulo $f(x)$ with coefficients in $\mathbb{Z}_q$. Addition in these rings is done component-wise in their coefficients (thus, their additive group is isomorphic to $\mathbb{Z}^n$ and $\mathbb{Z}_q^n$ respectively). Multiplication is simply polynomial multiplication modulo $f(x)$ (and also q, in the case of the ring R_q).

Thus an element in R (or R_q) can be viewed as a degree n polynomial over $\mathbb{Z}$ (or $\mathbb{Z}_q$). One can represent such an element using the vector of its coefficients. For an element $a(x) = a_0 + a_1x + \ldots + a_{n-1}x^{n-1} \in R$, we let $\|a\|_\infty = \max|a_i|$ denote its ℓ_∞-norm.

The $\mathsf{RLWE}_{f,q,\chi}$ assumption is parameterized by an irreducible integer polynomial $f(x) \in \mathbb{Z}[x]$ of degree n (which defines the ring $R = \mathbb{Z}[x]/\langle f(x)\rangle$), a prime integer $q \in \mathbb{Z}$ and an error distribution χ over R, and is defined as follows. Let $s \overset{\$}{\leftarrow} R_q$ be a uniformly random ring element. The assumption is that given any polynomial number of samples of the form $(a_i, b_i = a_i \cdot s + e_i) \in (R_q)^2$, where a_i is uniformly random in R_q and e_i is drawn from the error distribution χ, the b_i's are computationally indistinguishable from uniform in R_q. As shown in [ACPS09,LPR10], this is equivalent to a variant where the secret is sampled from the noise distribution χ rather than being uniform in R_q. It is also easy to see that the assumption is equivalent to a variant where the noise e_i are multiples of some integer t that is relatively prime to q.

We consider the RLWE problem for specific choices of the polynomial $f(x)$ and the error distribution χ. Namely,

[6] If the cloud and the advertiser collude, then the cloud may be able to learn some information about whether the user likes the ad or not, which reveals information about his preferences. This constitutes a form of CCA attack, which might endanger the security of the FHE.

– We set $f(x)$ to be the cyclotomic polynomial $x^n + 1$ for n a power of two. In addition to many other useful properties, the fact that $f(x) = x^n + 1$ means that multiplication of ring elements does not increase their L^2-norm by too much (see Lemma 2 below).

– The error distribution χ is the discrete Gaussian distribution $D_{\mathbb{Z}^n,\sigma}$ for some $\sigma > 0$, defined by the probability density function

$$\forall \mathbf{e} \in \mathbb{Z}^n : \quad \Pr[\mathbf{e} \leftarrow D_{\mathbb{Z}^n,\sigma}] = \frac{e^{-\pi||\mathbf{e}||^2/\sigma^2}}{\sum_{\mathbf{e}\in\mathbb{Z}^n} e^{-\pi||\mathbf{e}||^2/\sigma^2}}$$

A sample from this distribution defines a polynomial $e(x) \in R$.

We present some elementary facts about the Gaussian distribution, and multiplication over the ring $\mathbb{Z}[x]/\langle x^n + 1\rangle$. The first fact bounds the (Euclidean and therefore, the ℓ_∞) length of a vector drawn from a discrete Gaussian of standard deviation σ by $\sigma\sqrt{n}$. The second fact says that multiplication in the ring $\mathbb{Z}[x]/\langle x^n + 1\rangle$ increases the norm of the constituent elements only by a modest amount.

Lemma 1 (see [MR07], Theorem 4.4). *Let $n \in \mathbb{N}$. For any real number $\sigma > \omega(\sqrt{\log n})$, we have*

$$\Pr_{\mathbf{x}\leftarrow D_{\mathbb{Z}^n,\sigma}}[\|\mathbf{x}\|_\infty > \sigma\sqrt{n}] \leq 2^{-n+1}.$$

Lemma 2 (see [LM06,Gen09]). *Let $n \in \mathbb{N}$, and let $f(x) = x^n + 1$ and let $R = \mathbb{Z}[x]/\langle x^n + 1\rangle$. For any $s, t \in R$,*

$$||s \cdot t \pmod{x^n + 1}||_\infty \leq n \cdot ||s||_\infty \cdot ||t||_\infty .$$

Solving the RLWE problem (for the stated parameters) is also known to give us a quantum algorithm that solves short vector problems on ideal lattices with related parameters. The latter problem is believed to be exponentially hard.

3.2 Somewhat Homomorphic Encryption

The somewhat homomorphic encryption scheme

$$\mathcal{SHE} = (\mathsf{SH.Keygen}, \mathsf{SH.Enc}, \mathsf{SH.Add}, \mathsf{SH.Mult}, \mathsf{SH.Dec})$$

is associated with a number of parameters:

– the dimension n, which is a power of 2,
– the cyclotomic polynomial $f(x) = x^n + 1$,
– the modulus q, which is a prime such that $q \equiv 1 \pmod{2n}$,
  Together, n, q and $f(x)$ define rings $R := \mathbb{Z}[x]/\langle f(x)\rangle$ and $R_q := R/qR = \mathbb{Z}_q[x]/\langle f(x)\rangle$.
– the error parameter σ, which defines a discrete Gaussian error distribution $\chi = D_{\mathbb{Z}^n,\sigma}$ with standard deviation σ,
– a prime $t < q$, which defines the message space of the scheme as $R_t = \mathbb{Z}_t[x]/\langle f(x)\rangle$, the ring of integer polynomials modulo $f(x)$ and t, and
– a number $D > 0$, which defines a bound on the maximum number of multiplications that can be performed correctly using the scheme.

These parameters will be chosen (depending on the security parameter κ) in such a way as to guarantee correctness and security of the scheme. See Section 5 for concrete choices of parameters.

The Scheme SH.Keygen(1^κ): Sample a ring element $s \xleftarrow{\$} \chi$ and define the *secret key* $sk := s$. Sample a uniformly random ring element $a_1 \leftarrow R_q$ and an error $e \leftarrow \chi$ and compute the *public key* $pk := (a_0 = -(a_1 s + te), a_1)$.

Publish pk and keep sk secret.

SH.Enc(pk, m): Recall that our message space is R_t. That is, we encode our message as a degree n polynomial with coefficients in $\mathbb{Z}_t$.

Given the public key $pk = (a_0, a_1)$ and a message $m \in R_q$, the encryption algorithm samples $u \leftarrow \chi$, and $f, g \leftarrow \chi$, and computes the ciphertext

$$\mathsf{ct} = (c_0, c_1) := (a_0 u + tg + m, a_1 u + tf)$$

SH.Dec$(sk, \mathsf{ct} = (c_0, c_1, \ldots, c_\delta))$: To decrypt, we first compute

$$\widetilde{m} = \sum_{i=0}^{\delta} c_i s^i \in R_q$$

and output the message as $\widetilde{m} \pmod{t}$.

Homomorphic Operations. We now show how to compute the addition and multiplication operations homomorphically. To compute an arbitrary function f homomorphically, we construct an arithmetic circuit for f (made of addition and multiplication operations over $\mathbb{Z}_t$), and then use $\mathsf{SH.Add}$ and $\mathsf{SH.Mult}$ to iteratively compute f on encrypted inputs.

Although a ciphertext produced by $\mathsf{SH.Enc}$ contains two ring elements, the homomorphic operations (in particular, multiplication) increase the number of ring elements in the ciphertext. In general, the $\mathsf{SH.Add}$ and $\mathsf{SH.Mult}$ operations get as input two ciphertexts $\mathsf{ct} = (c_0, c_1, \ldots, c_\delta)$ and $\mathsf{ct}' = (c'_0, c'_1, \ldots, c'_\gamma)$. The output of $\mathsf{SH.Add}$ contains $\max(\delta+1, \gamma+1)$ ring elements, whereas the output of $\mathsf{SH.Mult}$ contains $\delta + \gamma + 1$ ring elements.

SH.Add$(pk, \mathsf{ct}_0, \mathsf{ct}_1)$: Let $\mathsf{ct} = (c_0, c_1, \ldots, c_\delta)$ and $\mathsf{ct}' = (c'_0, c'_1, \ldots, c'_\gamma)$ be the two ciphertexts (If $\delta \neq \gamma$, we pad the shorter ciphertext with zeroes).

Homomorphic addition is done by simple component-wise addition of the ciphertexts. Namely, compute and output

$$\mathsf{ct}_{\mathsf{add}} = (c_0 + c'_0, c_1 + c'_1, \ldots, c_{\max(\delta,\gamma)} + c'_{\max(\delta,\gamma)}) \in R_q^{\max(\delta,\gamma)+1}$$

SH.Mult$(pk, \mathsf{ct}_0, \mathsf{ct}_1)$: Let $\mathsf{ct} = (c_0, c_1, \ldots, c_\delta)$ and $\mathsf{ct}' = (c'_0, c'_1, \ldots, c'_\gamma)$ be the two ciphertexts. Here, we do not pad either of the ciphertexts with zeroes.

Let v be a *symbolic* variable and consider the expression

$$(\sum_{i=0}^{\delta} c_i v^i) \cdot (\sum_{i=0}^{\gamma} c'_i v^i)$$

(over R_q). We can (symbolically, treating v as an unknown variable) open the parentheses to compute $\hat{c}_0, \ldots, \hat{c}_{\delta+\gamma} \in R_q$ such that

$$\left(\sum_{i=0}^{\delta} c_i v^i\right) \cdot \left(\sum_{i=0}^{\gamma} c'_i v^i\right) \equiv \sum_{i=0}^{\delta+\gamma} \hat{c}_i v^i \ . \qquad (1)$$

The output ciphertext is $\mathsf{ct}_{\mathsf{mlt}} = (\hat{c}_0, \ldots, \hat{c}_{\delta+\gamma})$.

This scheme is correct, and secure under the Ring LWE assumption, as shown in [BV11b]. We state their theorem below, which also serves as the setting of the modulus q (in terms of σ, t, n and D) that ensures that the scheme can perform D multiplications and A additions.

Theorem 3. *The encryption scheme $\mathcal{SHE}$ is correct, and can compute D multiplications followed by A additions, assuming that*

$$q \geq 4 \cdot (2t\sigma^2\sqrt{n})^{D+1} \cdot (2n)^{D/2} \cdot \sqrt{A} \tag{2}$$

The encryption scheme $\mathcal{SHE}$ is secure under the Ring LWE assumption with parameters n, q and χ.

An Optimization to Reduce Ciphertext Size The homomorphic multiplication operation increases the number of ring elements in a ciphertext. Brakerski and Vaikuntanathan [BV11a] describe a transformation – called relinearization – that reduces the ciphertext back to two ring elements. We describe this optimization below, implement it and report on the performance numbers in Section 5.

Essentially, the idea is the following: assume that we run $\mathsf{SH.Mult}$ on two ciphertexts (each containing two ring elements) produced by the encryption algorithm. The resulting ciphertext $\mathsf{ct_{mlt}}$ contains three ring elements that satisfy the "invariant"

$$f_{\mathsf{ct_{mlt}}}(s) = c_2s^2 + c_1s + c_0 = te_{\mathsf{mult}} + mm'$$

This is a quadratic equation in s, and thus, $\mathsf{SH.Mult}$ turned two linear ciphertexts into a quadratic ciphertexts. The goal of *relinearization* is to bring this back down to a linear ciphertext.

To this end, we publish some "homomorphism keys" to aid relinearization. This could be thought of as part of the public key, but the homomorphism key is only used for relinearization (following an $\mathsf{SH.Mult}$ operation). The homomorphism key $hk = (h_1, \ldots, h_{\lceil \log_t q \rceil - 1})$ is computed as:

$$h_i = (a_i, b_i = -(a_is + te_i) + t^is^2) \quad \text{for } i = 0, \ldots, \lceil \log_t q \rceil - 1$$

where $a_i \leftarrow R_q$ and $e_i \leftarrow \chi$ are chosen independently for every i. In a sense, these are "quasi-encryptions" of $t^i \cdot s^2$. They are not real encryptions since $t^i \cdot s^2$ may not lie in the message space of the encryption scheme, namely R_t.

The homomorphic multiplication generates a ciphertext $\mathsf{ct_{mlt}} = (c_0, c_1, c_2)$, starting from two 2-element ciphertexts. Relinearization is performed after every homomorphic multiplication, and proceeds as follows.

1. Write the polynomial c_2 in its base-t representation as follows. $c_2 = \sum c_{2,i}t^i$ (for $i = 0, \ldots, \lceil \log_t q \rceil - 1$), where all the coefficients of $c_{2,i}$ are smaller than t.
2. Now, set

$$c_1^{\mathsf{relin}} := c_1 + \sum_{i=0}^{\lceil \log_t q \rceil - 1} c_{2,i}a_i \quad \text{and} \tag{3}$$

$$c_0^{\mathsf{relin}} := c_0 + \sum_{i=0}^{\lceil \log_t q \rceil - 1} c_{2,i}b_i \tag{4}$$

   where $h_i = (a_i, b_i)$ come from the "homomorphism key".
3. Output the 2-element ciphertext $\mathsf{ct_{mlt}} := (c_0^{\mathsf{relin}}, c_1^{\mathsf{relin}})$.

To see why this works, note that[7]

$$\begin{aligned} c_0^{\mathsf{relin}} &= c_0 + \sum_i c_{2,i}b_i \\ &= c_0 + \sum_i c_{2,i}(-a_is - te_i + t^is^2) \\ &= c_0 - \left(\sum_i c_{2,i}a_i\right)s - te_{\mathsf{relin}} + \left(\sum_i c_{2,i}t^i\right)s^2 \\ &= c_0 - (c_1^{\mathsf{relin}} - c_1)s - te_{\mathsf{relin}} + c_2s^2 \end{aligned}$$

[7] The summation runs from $i = 0$ to $i = \lceil \log_t q \rceil - 1$. We omit the indices for brevity.

from Equation 3 and by the definition of $c_2 := \sum_i c_{2,i} t^i$.

This means that

$$c_0^{\mathsf{relin}} + c_1^{\mathsf{relin}} s = c_0 + c_1 s + c_2 s^2 - t e_{\mathsf{relin}}$$

But, since $c_0 + c_1 s + c_2 s^2 = t e_{\mathsf{mult}} + mm'$, we have

$$c_0^{\mathsf{relin}} + c_1^{\mathsf{relin}} s = t(e_{\mathsf{mult}} - e_{\mathsf{relin}}) + mm'$$

thus maintaining the invariant and achieving correctness of decryption if the final error $e_{\mathsf{mult}} - e_{\mathsf{relin}}$ is small enough. Note that the relinearization process adds a fixed amount of error to the ciphertext, and does not accumulate error multiplicatively.

On the one hand, relinearization reduces the length of the ciphertext considerably. On the flip side, the public parameters become much larger. They now consist of an additional $\log_t q$ ring elements, totaling to $\log_t q \cdot n \lg q = n(\lg q)^2 / \lg t$ bits. Relinearization also affects the running time of the homomorphic multiplication. In particular, one needs to additionally perform roughly $\log_t q$ polynomial multiplications and additions. These (side-)effects are most pronounced for small t, where our experiments indicate considerable overhead. Quite encouragingly, though, the benefits of relinearization seem to dominate the side-effects for large t (see Section 5 for more details).

The security of the encryption scheme given the homomorphism keys relies on the circular security of the encryption scheme when encrypting quadratic functions of the secret key.

4 Message Encoding Techniques

The ease of performing homomorphic operations depends crucially on the specific message-encoding used in the ciphertexts. Consider the following two examples.

- If we wish to *compare two encrypted integers* $x, y \in \mathbb{Z}_t$ homomorphically, then it seems best to encrypt them *bit-wise* rather than as elements of $\mathbb{Z}_t$. The former approach translates to computing a polynomial of degree $\lg t$ over the encrypted bits, whereas the latter seems to require a polynomial of degree $O(t)$.
- If we wish to *compute the mean* of k integers, then it seems most natural to encode them as elements of $\mathbb{Z}_t$ (for a large enough t). Computing the mean homomorphically then involves only cheap homomorphic additions over $\mathbb{Z}_t$. On the other hand, if the numbers are encrypted bit-wise, then addition requires computation of expensive carry operations that involve homomorphic multiplication over $\mathbb{Z}_2$.

We describe two tricks for encoding messages. The first trick shows how to efficiently encode integers in a ciphertext so as to enable efficient computation of their sums and products *over the integers*. This is useful in computing the mean, the standard deviation and other private statistics efficiently. The second trick shows how to "pack" n encryptions of bits into a single encryption of the n-bit string. Some homomorphic operations, e.g., comparison of integers or private information retrieval, seem to require bit-wise encryptions of the input. Once the answers are computed, though, they can be packed into a single encryption using this trick.

4.1 Efficient Encoding of Integers for Arithmetic Operations

Given a list of integers $(m_1, \ldots, m_\ell) \in \mathbb{Z}^\ell$, if our goal is to compute their sum or product over the integers homomorphically, the obvious (and sub-optimal) choice is to encrypt them directly. Namely, for every m in the list, compute

$$\mathsf{Enc}(pk, m) = (c_0, c_1) = (a_0 u + tg + m, a_1 u + tf)$$

To ensure that we obtain $\sum_i m_i$ over the integers (and not mod t), we are forced to choose t to be rather large, namely $t > \sum_i m_i$, which could be rather prohibitive.

We show a method of encrypting integers more efficiently by encoding them in the polynomial ring, in essence enabling a smaller choice of t and better efficiency. In particular, for small enough $m_i < 2^n$, we show that it suffices to choose $t > \ell$ in order to add ℓ integers. Being able to work with a small t in turn enables us to choose other parameters, e.g., q and n to be correspondingly smaller.

The idea is very simple: break each m into (at most n) bits $(m^{(0)}, \ldots, m^{(n-1)})$, create a degree-(n-1) polynomial $\mathsf{pm}(x) = \sum_j m^{(j)} x^j$ and encrypt m as

$$\mathsf{Enc}(pk, m) = (c_0, c_1) = (a_0 u + tg + \mathsf{pm}, a_1 u + tf)$$

Adding these encryptions adds up the polynomials $\mathsf{pm}_i(x)$ *coefficient-wise.* Note that each coefficient was a single bit to start with, and a sum ℓ of them grows to at most ℓ. As long as $q > t > \ell$, this does not wrap around modulo t and upon decryption, we in fact get the polynomial $\mathsf{pm}_{\mathsf{add}}(x) = \sum_i \mathsf{pm}_i(x)$ over $\mathbb{Z}[x]$. Now, the result is simply $\mathsf{pm}_{\mathsf{add}}(2)$.

Extending this idea to support multiplication is a bit trickier. The problem stems from the fact that the product of two polynomials $\mathsf{pm}(x)$ and $\mathsf{pm}'(x)$ in general has a larger degree than each of the original polynomials. If their original degree was close to n to start with, we will only be able to obtain $\mathsf{pm}(x)\mathsf{pm}'(x) \pmod{x^n + 1}$ upon decryption, which loses information about the product. The solution is to encode the messages m as polynomials of degree at most n/d, if we anticipate performing d multiplications. For our applications (e.g., computing standard deviations), this is an acceptable trade-off since we only anticipate doing a single multiplication (or, at most a small number of them in the case of computing higher-order regression functions).

4.2 Packing Many Bits in a Ciphertext

We show how to transform ciphertexts that encode n bits $b_0, b_1, \ldots, b_{n-1}$ separately, into a single ciphertext that encodes the polynomial $b(x) = b_0 + b_1 x + \ldots + b_{n-1} x^{n-1}$.

Given n ciphertexts $\mathsf{ct}_i = (c_{0,i}, c_{1,i})$ that encrypt the bits b_i, it is easy to see that the ciphertext

$$\mathsf{ct}_{\mathsf{pack}} := (\sum_i c_{0,i} x^i, \sum_i c_{1,i} x^i)$$

encrypts the polynomial $b(x) = b_0 + \ldots + b_{n-1} x^{n-1}$. (It is equally easy to do this with homomorphically evaluated – and thus, potentially longer – ciphertexts as well).

In contrast, it seems much harder to *unpack* a ciphertext. Namely, transform a ciphertext that encodes the polynomial $b(x) = b_0 + \ldots + b_{n-1} x^{n-1}$ into n separate ciphertexts that encode the bits b_i. This is a useful thing to do when the homomorphic computation demands that the messages be encrypted bit-wise, forcing the client to send many ciphertexts, one for each bit. If we had a technique for unpacking bits, we could have the client send a single ciphertext, unpack it at the server's end, have the server perform computations, and finally, pack the result into one ciphertext to send it back.

5 Implementation Details

We have implemented the somewhat homomorphic public key encryption scheme in the computer algebra system MAGMA [BCP97] and ran experiments on an Intel Core 2 Duo processor at 2.1 GHz. We use MAGMA's polynomial arithmetic for all computations in R_q, in particular we use MAGMA's addition and multiplication of polynomials over $\mathbb{Z}_q$ modulo $x^n + 1$.

5.1 Choice of Parameters

We assess the security of our encryption scheme against the attacks described in [MR09,LP11]. We follow the analysis described in [LP11] and adjust it to our setting. In particular, this lets us compute secure parameter choices for the scheme, in order to support homomorphic computation of (multi-variate) polynomials of different degrees D. The results are summarized in Table 1. According to the analysis in [LP11], the chosen parameters provide a security level of 128 bits or more against the distinguishing attack with advantage $\epsilon = 2^{-32}$.

We now briefly explain the analysis of security against the distinguishing attack of [MR09]. We remark that the parameters in the tables also offer a large degree of protection against the more powerful decoding attack of [LP11]. For more details on these attacks, the reader is referred to the original papers [MR09,LP11].

In order to succeed with advantage ϵ, the distinguishing attack first produces vectors of length $c \cdot q/s$ in the dual $\Lambda^{\perp}(\mathbf{A})$, where $c \approx \sqrt{\lg(1/\epsilon)/(\lg 2 \cdot \pi)}$. This translates to $c \approx 2.657$ for $\epsilon = 2^{-32}$ or $c \approx 3.758$ for $\epsilon = 2^{-64}$.

The short vectors in $\Lambda^{\perp}(\mathbf{A})$ are produced using the BKZ algorithm, whose running time is determined by the root Hermite factor δ defined by the following equation:

$$c \cdot q/s = \delta^m \cdot \mathsf{det}(\Lambda^{\perp}(\mathbf{A}))^{1/m} = \delta^m \cdot q^{n/m}.$$

The optimal value of δ (and thus, the optimal running time of the attack) is achieved for $m = \sqrt{n \log q / \log \delta}$ which gives us the following relation between q, n and δ:

$$c \cdot q/s = 2^{2\sqrt{n \log q \log \delta}}. \tag{5}$$

To determine specific parameters, we fixed a value for n which we require to be a power of 2. The correctness condition (2) (or an experimentally confirmed smaller value) gives us a lower bound on the prime q. Fixing the prime q now allows us to solve equation (5) for δ. Using the heuristic run-time estimate of the attack provided by the equation

$$\lg t_{\mathsf{Adv}} = 1.8/\lg \delta - 110.$$

from [LP11] lets us determine the running time of attacks for various parameter settings.

5.2 Analysis of the Tables

Table 1 shows the choices of parameters n (the dimension) and $\log q$ (the bit-length of the modulus) for different values of t (the message space modulus) and D (the maximal degree for the polynomial whose homomorphic evaluation we can support). For each such n and $\log q$, we provide the security level achieved as the logarithm of the running time required for the distinguishing attack with distinguishing advantage $\epsilon = 2^{-32}$. To the extent possible, we have set the parameters n and $\log q$ so that the running time of the distinguishing attack is more than 2^{128} basic operations (although, see below for an explanation of apparent "jumps" in the running times in Table 1). All these calculations assume that the noise in ring LWE is chosen from a discrete Gaussian with standard deviation parameter $s = 8$.

In Table 1, we go on to show the sizes of the ciphertext and the public key for these values of n and $\log q$. The ciphertext and the public key in the scheme have two ring elements each. The "evaluation key" (required for degree reduction during homomorphic multiplication), on the other hand, has $\log_t q$ ring elements. In the last three columns in Table 1, we show the size of a single element of $\mathbb{R}_q$, the size of a ciphertext (that encrypts an element of R_t) and the size of the public key (including the evaluation key), respectively. We choose to include the evaluation key in calculating the size of the public key in order to illustrate the extent to which it increases the length of the (otherwise rather short) public key.

Our calculations show that both n and $\log q$ grow almost linearly in the maximal ciphertext degree D (more precisely, they grow as $D \log D$). Thus, the bit length of the ciphertext grows quadratically in D and that of the public key (including the evaluation key) grows cubically.[8]

A phenomenon worth explaining is that of discontinuity in the security levels in Table 1. To illustrate this, lets look at the two rows in Table 1 corresponding to $t = 2$ and $D = 4$. Since our

[8] In the graph in Figure 1, it seems as if the growth of the ciphertext is linear and that of the public key is quadratic, but this is just an artifact of the limited parameter settings we implement.

system requires the polynomial $f(x)$ to be cyclotomic of form x^n+1, we are constrained to choose n to be a power of two. In the first of the two rows, we set $n = 2048$ and in the second, we set $n = 4096$, the next possible choice. The security, measured as the logarithm of the running time of the distinguishing attack, jumps from $\lg(T) = 64$ to $\lg(T) = 218$ when n goes from 2048 to 4096. In other words, suppose we require 128 bits of security for the system, we are constrained to one of two choices – set $n = 2048$ (and risk the possibility of better attacks), or set $n = 4096$ (and settle for worse efficiency). This conundrum arises due to the constraint on n being a power of two, which can potentially be removed by working with general cyclotomics (i.e., not of the form x^n+1). Smart and Vercauteren [SV11] provide additional motivations for such generalization.

In Table 2, we provide running times for key generation, encryption, decryption and homomorphic addition and multiplication for various parameters. For $\mathsf{SH.Enc}$, we distinguish the case where sampling from the discrete Gaussian χ is included in the measurement and the case where it is done as precomputation. For $\mathsf{SH.Dec}$ we give timings for decrypting ciphertexts of degree 1 and of degree 2. The two columns with timings for $\mathsf{SH.Mult}$ correspond to multiplication of degree-1 ciphertexts with and without relinearization.

Mean and variance computation. To compute the mean, we do not need any multiplications, just additions of ciphertexts, i.e. the maximal degree of ciphertext we need is $D = 1$. We used the parameters from Table 1 with $t = 1024$, $D = 1$ and $n = 1024$. The corresponding 30-bit prime is $q = 1061093377$ and has been chosen so as to support up to 1000 additions. We do not compute the ciphertext of the mean, but of the sum of all numbers instead together with a ciphertext encrypting the number of numbers that have been added. The mean can then easily be computed by one division after decryption. Computing the ciphertext for the sum of 100 numbers of size 128-bits from the single ciphertexts takes about 20ms.

Computation of the variance requires one multiplication. Suitable parameters are given in Table 1 as $t = 1024$, $D = 2$, and $n = 2048$ with a 58-bit prime $q = 144115188076060673$. To obtain the ciphertexts for the sum and sum of squares that can be used to determine mean and variance takes about 6s.

Potential Improvements. We remark that our implementation uses the generic polynomial arithmetic in MAGMA. A number of performance optimizations are possible; we mention one such possibility. The encryption scheme uses addition and multiplication of polynomials over $\mathbb{Z}_q$ modulo x^n+1, where n is a power of two and $q = 1 \pmod{2n}$. However, the particular choice of n and q could allow for much faster implementations than the generic MAGMA code. Such optimizations have already been considered in the context of hash functions (e.g., SWIFFT [LMPR08]) that use fast Fourier-transform techniques to speed up computations.

6 Extensions and Future Work

Implementing Fully Homomorphic Encryption. The somewhat homomorphic encryption scheme of [BV11b] can be turned into a fully homomorphic encryption scheme using the relinearization and the dimension reduction techniques of [BV11a]. We leave the problem of implementing the resulting fully homomorphic encryption scheme (along the lines of [GH11,SV11] who implement Gentry's FHE scheme) as an important future work. Implementing the [BV11a] FHE could also lead to a number of nice applications, for example, to the problem of optimizing communication with the cloud described below.

Optimizing communication with the cloud. We present a solution to help mitigate the problem of the large ciphertext size for the Ring-LWE based FHE solution. In any of the above applications, a client communicates with the cloud service and uploads its data encrypted under a FHE scheme, and the cloud operates on this data and returns encrypted outputs to the client. Each ciphertext has size $n\log(q)$, and for functions requiring a large number of multiplications, q and n could be very large (see the implementation section for sample choices of q and n).

The solution to this is two fold. First, all encryptions that the client sends to the server can be encrypted using (a semantically secure encryption scheme based on) AES (which, by itself, is not homomorphic at all). The main observation is that the steps of AES decryption can all be carried out on FHE-encrypted entries. A one time set-up cost is that the client uploads the FHE-encryption of its AES secret key K:

$$\text{"Client sends } \mathsf{FHE}(K) \text{ to Cloud"}$$

Then for each piece of content m to be uploaded to the cloud, the client uploads only the AES-encryption of m to the cloud, encrypted under its own secret key K.

$$\text{"Client sends } \mathsf{AES}_K(m) \text{ to Cloud"}$$

Now the cloud is expected to operate on the inputs it receives for the client and compute and return FHE-encryptions of functions of those inputs. In order to do that, the cloud must first compute the FHE-encryption of m: in other words the cloud computes the public key FHE-encryption of the AES encryption of the content, $\mathsf{FHE}(\mathsf{AES}_K(m))$, and must now unravel the AES encryption inside the FHE encryption to obtain $\mathsf{FHE}(m)$. Once this is done, the cloud computes the FHE encryption of $f(m)$, for the appropriate function f.

There is still a snag in this solution, namely that the resulting ciphertext that the server returns to the client is still a large FHE ciphertext. The solution to this is the dimension reduction technique introduced by [BV11a]. In particular, the dimension reduction technique converts a ciphertext in $\mathbb{Z}_q[x]/\langle x^n + 1\rangle$ (where both n and q are large in order to support expressive homomorphisms) to a ciphertext in $\mathbb{Z}_p[x]/\langle x^k + 1\rangle$, where both k and p are small. The resulting ciphertext encrypts the same message, although it does not support any further homomorphisms. The server then applies this transformation and sends the resulting short ciphertext to the client.

In short, all the communication over the network consists of short, non-homomorphic ciphertexts. At the server's end, the ciphertexts are first "upgraded" to homomorphic ciphertexts which are then computed on, and finally "downgraded" to short non-homomorphic ciphertexts which are then sent to the client.

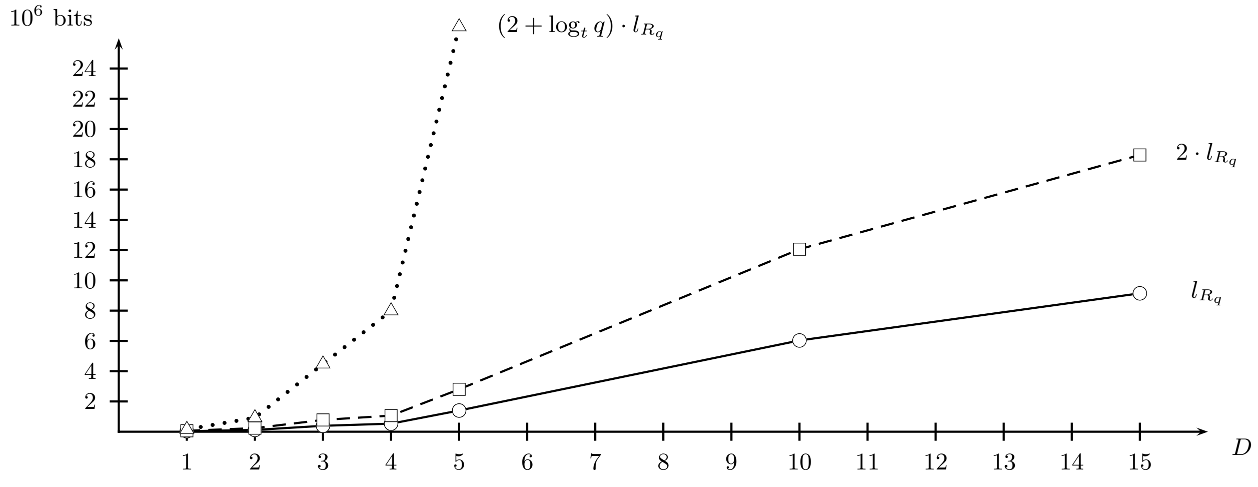

Fig. 1. Graphical representation of part of the data in the last three columns of Table 1 for $t = 1024$, giving the size l_{R_q} of a general element in R_q, public key size, original ciphertext size (both $2 \cdot l_{R_q}$) and public key size including the parameters for the relinearization step.

7 Conclusion

In answer to the question of whether homomorphic encryption can be practical, we give several concrete applications to useful cloud computing scenarios where only somewhat homomorphic en-

t	D	n	$\lceil \lg(q) \rceil$	δ	lg(T)	$l_{R_q}/10^3$	$2 \cdot l_{R_q}/10^3$	$(2 + \log_t q) \cdot l_{R_q}/10^3$
2	1	512	19	1.0054	123	10	20	205
	2	1024	38	1.0058	107	39	78	1557
	3	2048	64	1.0051	134	132	263	8651
	4	2048	89	1.0072	64	183	365	16587
	4	4096	94	1.0038	218	386	771	36963
	5	4096	120	1.0049	145	492	984	59966
	10	8192	264	1.0055	117	2163	4326	575276
	15	16384	423	1.0044	172	6931	13861	2945434
128	1	1024	27	1.0041	199	28	56	162
	2	2048	52	1.0041	198	107	213	1005
	3	2048	82	1.0067	78	168	336	2304
	3	4096	86	1.0035	250	353	705	5033
	4	4096	118	1.0048	149	484	967	9115
	5	4096	150	1.0062	92	615	1229	14395
	10	8192	324	1.0068	74	2655	5309	128161
	10	16384	338	1.0035	243	5538	11076	278472
	15	16384	513	1.0054	122	8405	16810	632776
1024	1	1024	30	1.0047	164	31	62	154
	2	2048	58	1.0046	164	119	238	927
	3	2048	91	1.0074	59	187	373	2069
	3	4096	95	1.0039	215	390	779	4475
	4	4096	130	1.0053	124	533	1065	7988
	5	4096	165	1.0068	73	676	1352	12504
	5	8192	171	1.0035	242	1401	2802	26756
	10	8192	354	1.0074	59	2900	5800	108459
	10	16384	368	1.0039	214	6030	12059	233938
	15	16384	558	1.0059	103	9143	18285	528424
	32	65536	1298	1.0034	255	85066	170132	11211663
	64	131072	2705	1.0036	239	354550	709100	96614810

Table 1. Example parameters and cost of the distinguishing attack from [LP11] for distinguishing advantage $\epsilon = 2^{-32}$, i.e. $c \approx 2.657$, modulus t for the message space R_t, maximal ciphertext degree D, size of prime q, Hermite root factor δ, and logarithm of the runtime $\lg(T)$. We also give the bitlength $l_{R_q} = n \cdot \lceil \lg(q) \rceil$ of an element from R_q and the bitlength $2 \cdot l_{R_q}$ of the public key and an original ciphertext that both consist of 2 elements. The last column gives the size of the public key together with the $\log_t q$ public elements used for the relinearization technique described in Section 3.2. The entries in the last three columns are given in multiples of 10^3 bits and are rounded up to the next such multiple.

				S_χ	SH.Keygen	SH.Enc		SH.Dec		SH.Add	SH.Mult	SH.Mult
							precomp.	deg 1	deg 2			w/ deg red
t	D	n	$\lceil \lg(q) \rceil$	ms	ms	ms	ms	ms	ms	ms	ms	s
2	1	512	19	27	60	81	2	2	–	< 1	–	–
	2	1024	38	55	120	171	9	6	10	1	15	0.34
	3	2048	64	110	260	353	29	18	33	1	56	1.98
	4	2048	89	111	270	357	32	19	35	1	59	2.94
	4	4096	94	221	540	733	82	46	89	2	155	7.63
	5	4096	120	223	560	742	85	49	94	3	163	10.59
	10	8192	264	438	1480	1738	425	227	454	7	887	114.57
	15	16384	423	880	4000	4176	1503	781	1561	14	3160	669.40
128	1	1024	27	54	110	163	4	4	–	< 1	–	–
	2	2048	52	110	270	348	23	15	25	1	41	0.23
	3	2048	82	110	270	357	32	20	35	1	60	0.44
	3	4096	86	222	520	724	69	41	77	4	130	1.05
	4	4096	118	221	550	740	86	49	93	4	162	1.62
	5	4096	150	221	590	771	117	65	124	4	226	2.76
	10	8192	324	437	1620	1845	548	283	565	6	1069	26.17
	10	16384	338	870	3540	3864	1269	656	1327	19	2501	63.49
	15	16384	513	864	4710	4503	1925	977	1960	29	3844	145.55
1024	1	1024	30	54	110	164	5	4	–	< 1	–	–
	2	2048	58	110	250	348	24	15	26	1	41	0.19
	3	2048	91	111	270	366	38	22	41	2	73	0.46
	3	4096	95	221	530	733	81	46	88	4	154	0.95
	4	4096	130	220	580	756	102	57	109	4	196	1.50
	5	4096	165	220	600	770	117	64	125	4	226	2.19
	5	8192	171	440	1250	1582	275	148	288	5	526	5.33
	10	8192	354	435	1720	1824	523	271	538	9	538	19.28
	10	16384	368	868	3690	3851	1260	664	1300	19	1593	48.23
	15	16384	558	863	5010	4805	2343	1136	2269	13	4411	126.25

Table 2. Timings for the somewhat homomorphic encryption scheme using the example parameters given in Table 1. The column labeled S_χ gives timing for sampling an element from the discrete Gaussian distribution χ. In the second column for SH.Enc, labeled prec., encryption is measured without sampling from χ, which is instead done as a precomputation. The two columns for SH.Dec correspond to decryption of a degree-1 and a degree-2 ciphertext, respectively. The last column gives the time taken for a ciphertext multiplication of two linear ciphertexts including the degree reduction resulting in a degree-1 ciphertext for the product. Measurements were done on a 2.1 GHz Intel Core 2 Duo using the computer algebra system Magma [BCP97].

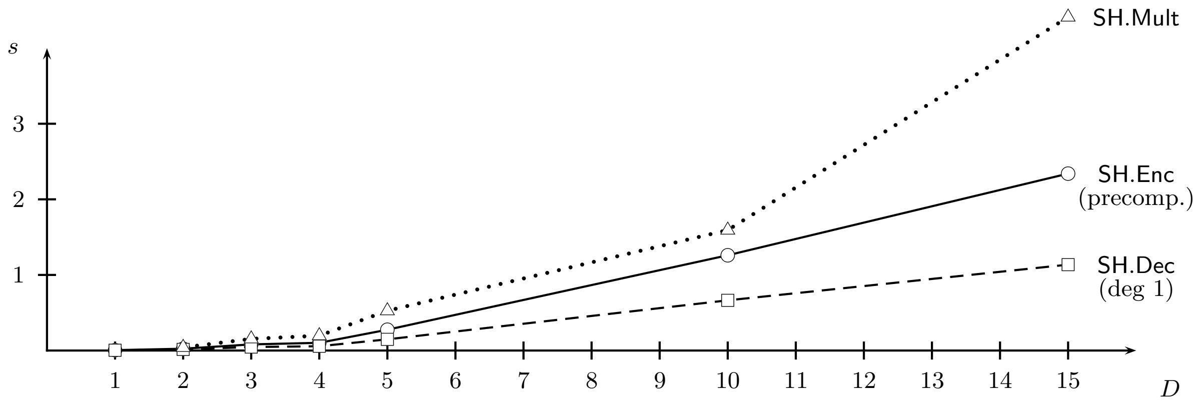

Fig. 2. Graphical representation of part of the timing data in Table 2 for $t = 1024$, giving the time for encryption SH.Enc when sampling from the discrete Gaussian distribution χ is done as a precomputation, the time for decryption SH.Dec of a ciphertext of degree 1, and the time for multiplication SH.Mult of two ciphertexts of degree 1.

cryption is required, and we give performance numbers based on our Magma implementation of the new scheme of [BV11b]. The scenarios we present are such that outsourced computation is actually valuable. That is, the cloud service may collect encrypted data from multiple sources and do computation on that data before providing useful encrypted content back to the user. Precise performance data was given in Section 5, but it is worthwhile to highlight here a sample of the results. We showed that an encryption for the sum of 100 128-bit numbers can be computed from the individual ciphertexts in 20 milliseconds on a laptop running Magma. We noted that computation of the variance requires one multiplication and taking suitable parameters from Table 1, we found that for $t = 1024, n = 2048$, and a 58-bit prime q, ciphertexts for the sum and sum of squares can be computed in about 6 seconds.

Acknowledgments. We would like to thank Daniele Micciancio for valuable comments.

References

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In Shai Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 595–618. Springer, 2009.

[BCHL09] Josh Benaloh, Melissa Chase, Eric Horvitz, Kristin Lauter. Patient-controlled encryption: patient privacy in electronic medical records. In *Proceedings of the 2009 ACM workshop on Cloud computing security.*

[BCP97] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system I: The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993).

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In *Theory of Cryptography - TCC'05*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341. Springer, 2005.

[BV11a] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. FOCS, 2011.

[BV11b] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. CRYPTO 2011, 2011.

[CF85] Josh D. Cohen and Michael J. Fischer. A robust and verifiable cryptographically secure election scheme (extended abstract). In *FOCS*, pages 372–382. IEEE, 1985.

[DGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In EUROCRYPT 2010, pages 24–43.

[El-84] Taher El-Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, pages 10–18, 1984.

[Fre10] David Mandell Freeman. Converting pairing-based cryptosystems from composite-order groups to prime-order groups. In EUROCRYPT 2010, pages 44–61.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *STOC*, pages 169–178. ACM, 2009.

[Gen10] Craig Gentry. Toward basing fully homomorphic encryption on worst-case hardness. In Tal Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 116–137. Springer, 2010.

[GH11] Craig Gentry and Shai Halevi. Implementing Gentry's fully-homomorphic encryption scheme. In *EUROCRYPT*, 2011. (To appear).

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *STOC*, pages 365–377. ACM, 1982.

[GN08] Nicolas Gama and Phong Q. Nguyen. Predicting lattice reduction. In Nigel P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 31–51. Springer, 2008.

[LM06] Vadim Lyubashevsky and Daniele Micciancio. Generalized compact knapsacks are collision resistant. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP (2)*, volume 4052 of *Lecture Notes in Computer Science*, pages 144–155. Springer, 2006.

[LMPR08] Vadim Lyubashevsky, Daniele Micciancio, Chris Peikert, and Alon Rosen. Swifft: A modest proposal for FFT hashing. In Kaisa Nyberg, editor, *FSE*, volume 5086 of *Lecture Notes in Computer Science*, pages 54–72. Springer, 2008.

[LP11] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In Aggelos Kiayias, editor, *CT-RSA*, volume 6558 of *Lecture Notes in Computer Science*, pages 319–339. Springer, 2011.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Eurocrypt 2010, pages 1–23. Draft of full version was provided by the authors.

[LV01] Arjen K. Lenstra and Eric R. Verheul. Selecting cryptographic key sizes. *J. Cryptology*, 14(4):255–293, 2001.

[MR07] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on gaussian measures. *SIAM J. Comput.*, 37(1):267–302, 2007.

[MR09] Daniele Micciancio and Oded Regev. Lattice-based cryptography. In *Post-Quantum Cryptography.* Springer, 2009.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[RS10] Markus Rückert and Michael Schneider. Estimating the security of lattice-based cryptosystems. Cryptology ePrint Archive, Report 2010/137, 2010. http://eprint.iacr.org/2010/137.

[SS10] Damien Stehlé and Ron Steinfeld. Faster fully homomorphic encryption. In Masayuki Abe, editor, *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 377–394. Springer, 2010.

[SV10] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In Phong Q. Nguyen and David Pointcheval, editors, *Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443. Springer, 2010.

[SV11] N.P. Smart and F. Vercauteren. Fully homomorphic SIMD operations. Cryptology ePrint Archive, Report 2011/133, 2011. http://eprint.iacr.org/2011/133.