Practical Private-key Fully Homomorphic Encryption in Rings

Alexey Gribov[1], Delaram Kahrobaei[1], and Vladimir Shpilrain[1]

City University of New York
gribov.alesha@gmail.com dkahrobaei@gc.cuny.edu
shpil@groups.sci.ccny.cuny.edu *

Abstract. We describe a practical fully homomorphic encryption (FHE) scheme based on homomorphisms between rings and show that it enables very efficient computation on encrypted data. Our encryption though is private-key; public information is only used to operate on encrypted data without decrypting it. Still, we show that our method allows for a third party search on encrypted data.

1 Introduction

The most widely known existing solution to the fully homomorphic encryption (FHE) problem is due to Craig Gentry [5], subsequently improved in [1] and other papers, and the relevant software has been developed by IBM [7]. This solution relies in its security on variants of the "bounded-distance decoding" problem that has the property of *random self-reducibility*, which basically means that it is about as hard on average as it is in the worst case. While this property is indeed a good evidence of security, the resulting homomorphic encryption algorithm is slow, and more importantly, computation on data encrypted with this algorithm is too inefficient to be practical. Very informally, the reason is that, to provide semantic security, encryption has to be randomized, but on the other hand, a homomorphism should map zero to zero. To resolve this conflict, the ciphertext zero is "masked" by "noise". The problem now is that during any computation on encrypted data, this "noise" tends to accumulate and has to be occasionally reduced by recryption (also known as *bootstrapping*), a process that produces the equivalent ciphertext but with less noise. Recryption is an expensive procedure, and this is what limits real-life applications of the existing FHE solutions. There were other proposals for fully homomorphic encryption following Gentry's, see e.g. [2], [4], [3] (the latter is actually very simple conceptually), but they all still involve "bootstrapping".

Our approach to homomorphic encryption is based on the use of rings (with usual operations of addition and multiplication) and homomorphisms between them. This has an additional benefit (as far as most real-life applications are concerned) of avoiding an "overhead" coming from converting real-life arithmetic to Boolean circuits and vice versa. Recall that a map $\varphi : R \to S$ between rings is a (full) homomorphism if $\varphi(u+v) = \varphi(u) + \varphi(v)$ and $\varphi(uv) = \varphi(u)\varphi(v)$ for any $u, v \in R$. This implies, in particular, that

* Research of all three authors was partially supported by the ONR (Office of Naval Research) grant N000141210758.

$\varphi(0) = 0$, i.e., a homomorphic encryption cannot possibly be semantically secure. Thus, when people use the expression "homomorphic encryption", what they actually mean is that the encryption function is a homomorphism modulo some ideal (whose elements are often called "noise") of the ring S. This ideal then gets "killed" during decryption, and therefore its elements do not influence the result of computation on plaintexts.

In Section 2, we give a high-level description of our FHE method, and in Section 4 we describe a practical instantiation of this general scheme. In particular, we show why computation on encrypted data is done without accumulating noise. In Section 3, we discuss security of our scheme.

Since our FHE scheme is private-key by design, its security properties are different from those of Gentry's and other similar public-key schemes. We give security overview in Section 3; here we just mention that our scheme is unconditionally (i.e., without any computational assumptions) secure against ciphertext-only attack, whereas the situation with known-plaintext attacks is more delicate and is addressed in conjunction with specific applications of FHE. Basically, we "sacrifice" (to some extent) encryptions of zero but protect privacy of nonzero elements. This can be considered a theoretical vulnerability, but this is not a problem for most real-life applications.

2 Encryption overview

We emphasize once again that our FHE scheme is private-key. This may somewhat limit the range of applications, but on the other hand, this leaves a lot of flexibility in terms of how much information the data owner is willing to give to the entity that stores (and operates on) encrypted data. In particular, this can provide a customer (a company that holds sensitive data) with a "toolbox" for building their own instantiation of our general FHE scheme, so that nobody outside the company would know details of the encryption/decryption mechanisms, even though the public should be able to operate on encrypted data. The latter requirement makes "security through obscurity" impossible for FHE schemes, so there should be at least some minimum amount of information available to the public, and this minimum is essentially just the general framework that we describe in this section.

1. Plaintexts are elements of a (private or public) ring R.
2. Ciphertexts are elements of a public ring S, such that $R \subset S$ is a subring of S. The ring S also has a (private) ideal I such that $S/I = R'$, where the ring R' is isomorphic to R. (The ring R' may be just equal to R, in which case R is called a *retract* of S.)
3. Given $u \in R$, encryption $E(u) = u + E(0)$, where $E(0)$ is a random element of the (private) ideal I of the ring S.
   Technically, this encryption function is not a homomorphism, but it is a homomorphism modulo the ideal I, which is what matters here. It obviously respects addition modulo I, and for multiplication we have: $E(u)E(v) = (u + j_1)(v + j_2) = uv + j_1u + uj_2 + j_1j_2 = uv + j_3 = E(uv)$ modulo I, where $j_1, j_2, j_3 \in I$.
4. Decryption is a map ρ from S to R' that takes every element of I to 0, followed by an isomorphism $\varphi : R' \to R$.

Here is a diagram to "visualize" this general scheme:

$$R \xrightarrow{E} S \xrightarrow{\rho} R' \xrightarrow{\varphi} R.$$

Note that when we say "a public ring S", this means that we give to the public a collection of rules for adding and multiplying elements of S. Typically, this can be a (finite) set of elements that generate S as a ring, together with the multiplication table for this set of elements.

Below is a diagram of the whole encryption process starting with a real-life database D:

$$D \xrightarrow{\alpha} R \xrightarrow{E} S \xrightarrow{\rho} R' \xrightarrow{\varphi} R \xrightarrow{\beta} D,$$

where $\beta(\alpha(x))) = x$ for any $x \in D$. All maps in this diagram are private, except that in some applications, α (and therefore also β) may be shared with a third party, see e.g. our Section 8.1.

We note again that the ring R can be private, although some general information about it is usually available to the public.

3 Security overview

Recall again that our encryption is *private-key*, and in particular, in the model considered in this paper, we do not give any access to the encryption mechanism to the third party.

We have to remind the reader, just in case, that there is a difference in security analysis between public-key and private-key encryption. In particular, "brute force" attacks in the public-key sense (e.g. encryption emulation attack) are usually not applicable if the encryption is private-key, unless plaintexts have some distinctive (usually non-mathematical) features, like words that "make sense". For example, if one encrypts a credit card number by, say, adding to it a random number modulo some p, then this encryption will be perfectly secure against encryption emulation attack, although completely insecure against known-plaintext attack (KPA).

That said, we point out that FHE is a rather special case: even if the encryption itself is private-key, some information has to be made public because otherwise, the public will not be able to operate on encrypted data, and this will defy the purpose of FHE. Still, we manage to take advantage of our encryption being private-key, and this allows us, in particular, to make our scheme unconditionally (i.e., without any computational assumptions) secure against ciphertext-only attack, which is not quite trivial for an FHE scheme even if it is private-key. As we pointed out in the previous paragraph, the real challenge though is to make a private-key scheme KPA secure. Since our encryption is fully homomorphic, of particular importance is the following question: does accumulating several known encryptions of 0 allow the adversary to decrypt without ambiguity? Or at least to recognize other encryptions of 0? The answer to the latter question is always "yes" because the ideal where encryptions of 0 live is finite-dimensional, so the question is only about *how many* encryptions of 0 are sufficient to know to find a linear basis of the ideal and therefore to be able to recognize other encryptions of 0.

In our scheme, this number is not large, but even if the adversary learns how to recognize encryptions of 0, this will not allow him to decrypt nonzero elements without ambiguity, which is what matters in most real-life applications. Still, in Section 7, we show how to prevent accumulation of encryptions of 0 during private search on encrypted database because we think this might be of independent interest.

In general, we bifurcate our scheme between two cases, depending on the nature of the data to be encrypted and on relevant real-life applications:

(1) Data do not have a ring structure; for example, if a database D consists of credit card numbers (or phone numbers, etc.), then the embedding of plaintexts in the ring R can be just one-to-one but not necessarily a full homomorphism. This is the most popular case as far as real-life applications of FHE are concerned.

(2) Data do have a ring structure, in which case embedding of a database D in the ring R of plaintexts has to be a full homomorphism, i.e., both additive and multiplicative, which inevitably makes encryption less secure. This case is relevant to the problem of delegating various computations to a remote server. This case is more delicate from the security point of view, but at the very least we have security against ciphertext-only attack, see Section 6.

4 The platform ring

We start by introducing a series of rings S_n:

$$S_n = \langle x_1, \ldots, x_n \mid p \cdot 1 = 0,\ x_i^2 = x_i,\ x_i x_j = x_j x_i\ (for\ all\ i, j) \rangle.$$

The first relation $p \cdot 1 = 0$ just means that the ring S_n is a linear algebra over $\mathbb{Z}_p$.

We note that S_n is a linear vector space over $\mathbb{Z}_p$. The "standard" basis of this vector space consists of 1 and monomials of the form $x_{i_1}, \ldots, x_{i_k}$ with $i_1 < i_2 < \ldots < i_k$. This is because the ring S_n is commutative and $x_i^2 = x_i$ for all i. The number of these monomials therefore equals the number of subsets of the set of generators, i.e., equals 2^n. It follows that the number of elements in S_n is p^{2^n}, so for example with $n = 7$ this will be p^{128}.

Computing in the ring S_n is very similar to computing with polynomials. Addition in S_n is just adding coefficients at the corresponding monomials. Multiplication amounts to multiplying out "polynomials" in x_i the usual way, followed by applying the relations $x_i^2 = x_i$. The latter amounts to just "erasing" all integers greater than 1 in the exponents. This allows one, in particular, to use known fast algorithms for multiplying usual polynomials, followed by "erasing" exponents greater than 1. Moreover, we show in the next Section 4.1 that the ring S_n has an *orthogonal* basis (as a linear space over $\mathbb{Z}_p$) consisting of idempotents e_i such that $e_i e_j = 0$ for all $i \neq j$, which means that S_n is isomorphic to a direct sum of 2^n copies of the ring $\mathbb{Z}_p$.

4.1 Idempotent elements and orthogonal basis

The ring S_n has many idempotents. From the relations $x_i^2 = x_i$ we see that each x_i is an idempotent, and since a product of idempotents is again an idempotent, every monomial

is an idempotent in S_n, so there are at least 2^n idempotents. There are also other idempotents in S_n, which are not monomials; for example, $g = x_i - x_i x_j + x_j$ is an idempotent if $i \neq j$. In fact, the number of idempotents in S_n is *superexponential* in n.

Indeed, for any set of indices $F \subset \{1,\dots,n\}$, denote $e_F := \prod_{i\in F} x_i \cdot \prod_{j\notin F}(1-x_j)$. It is easy to see that each e_F is an idempotent, and there are 2^n of them, by the number of subsets of $\{1,\dots,n\}$. Moreover, these idempotents e_F are pairwise *orthogonal*, i.e., $e_F e_G = 0$ if $F \neq G$. Therefore, any sum of different e_F is again an idempotent, and there are 2^{2^n} of such sums. For example, with $n = 7$, there are 2^{128} idempotents in the ring S_n.

We emphasize again (because this is important for computation) that these e_F form an *orthogonal* linear basis of S_n over $\mathbb{Z}_p$, which makes multiplication in S_n very efficient as long as elements of S_n are represented as linear combinations of e_F. We will therefore publish the ring S_r (for an appropriate r, see the next section) as follows:

$$S_r = \langle e_1,\dots,e_{2^r} \mid p\cdot 1 = 0,\ e_i^2 = e_i\ (for\ all\ i),\ e_i e_j = 0\ for\ i \neq j\rangle.$$

The order of e_i is random, and it is part of the private key.

Thus, encryption will include conversion of elements of S_r from the "standard" basis $\{x_i\}$ to the orthogonal basis $\{e_i\}$, and decryption will include conversion from the orthogonal basis to standard.

4.2 Selecting an ideal for encryptions of 0

We are now going to define a private ring $R = S_r/I$ and a private ideal I of the ring S that will be used for (randomized) encryptions of 0.

The ideal I is going to be generated (as an ideal of S_r) by several elements of the form $(x_m - w(x_1,\dots,x_{m-1}))$, for different m, where $m \geq n+1$ and $w = w(x_1,\dots,x_{m-1})$ is a random idempotent element of S_{m-1} (see our Section 4.1).

The ring R is the factor ring S_r/I, where r is the total number of x_i involved. This ring is naturally isomorphic to S_n. See our Section 5.1 for the recommended values of parameters.

5 Protocol

Now we give a more formal description of the encryption/decryption protocol. Note that in this protocol, elements of a *private* ring R are encrypted. The ring R is isomorphic to the factor ring of the public ring S_r by a private ideal I, and the latter factor ring is naturally isomorphic to S_n with $n < r$. For embedding elements of a real-life database in S_n, see the discussion below in Sections 5.2, 5.3. The protocol in this section is targeted mostly at the case where elements of the original database do not have a ring structure (e.g. credit card numbers, phone numbers, etc.), see also Section 5.2. The case where elements of the original database do have a ring structure requires an enhancement to the key generation part of the protocol. We explain this in Section 5.3 to avoid overloading the basic protocol below with technical details.

1. **Key generation I.** Alice (the owner of a private database) starts with a presentation of the ring S_n (with a private n):

$$S_n = \langle x_1, \ldots, x_n \mid p \cdot 1 = 0,\ x_i^2 = x_i,\ x_i x_j = x_j x_i\ (for\ all\ i, j) \rangle.$$

2. **Key generation II.** Alice generates a private encryption/decryption key as follows. She starts by expanding the set $\{x_i\}$ of the generators by adding several new generators $x_{n+1}, \ldots, x_r$. She then selects an ideal I of S_r generated (as an ideal of S_r) by elements of the form $(x_m - w_m(x_1, \ldots, x_{m-1}))$, for $m = n+1, \ldots, r$, where $w_m = w_m(x_1, \ldots, x_{m-1})$ is a random idempotent element of S_{m-1}.
3. **Key generation III.** Alice converts the basis $\{x_i\}$ to the orthogonal basis $\{e_i\}$, i.e., she represents each x_i as a linear combination of e_i. It may happen that all generators of the ideal I selected at the previous step have the same coordinates (in this orthogonal basis) equal to 0. These coordinates are then discarded by Alice, i.e., the public ring may have dimension (as a linear vector space over $\mathbb{Z}_p$) somewhat smaller than 2^r.

   After that, Alice selects a random permutation π on the set of remaining e_i and publishes a presentation P:

$$P = \langle e_1, \ldots, e_s \mid p \cdot 1 = 0,\ e_i^2 = e_i\ (for\ all\ i),\ e_i e_j = 0\ for\ i \neq j \rangle.$$

4. **Encryption.** Encryption of a plaintext $u \in S_n$ is $E(u) = u + E(0)$, where $E(0)$ is a random element of the ideal I of the ring S_r, i.e., an element of the form $\sum_{j=n+1}^{r} (x_j - w_j(x_1, \ldots, x_{j-1})) \cdot h_j(x_1, \ldots, x_r)$, where h_j are random elements of S_r, i.e., sums of monomials in $x_1, \ldots, x_r$ with random coefficients from $\mathbb{Z}_p$. The whole expression $E(u) = u + E(0)$ is then converted to a linear combination of e_i, where $\{e_i\}$ is the published orthogonal basis.
5. **Decryption.** To decrypt $g = g(e_1, \ldots, e_{2^r})$, Alice first converts g to the "standard" basis $\{x_i\}$. After that, she replaces x_j by $w_j(x_1, \ldots, x_{j-1})$, starting with $j = r$ and going down to $j = n+1$. The result is then an element of S_n.

5.1 Suggested parameters and description of the private key

The number p in $\mathbb{Z}_p$ can be on the order of 30 bits if plaintexts do not have a ring structure (e.g. credit card numbers, or names, etc.). If plaintexts do have a ring structure, p has to be on the order of t bits, where t is the security parameter, to prevent brute force attacks as explained in Remark 2 in Section 7.

The suggested values of n are in the range from 3 to 9; in our tests we used $n = 7$, in which case S_n has dimension 128 as a linear vector space over $\mathbb{Z}_p$. The suggested value of r is $n+3$.

The private key consists of:

(1) The generators $(x_m - w_m(x_1, \ldots, x_{m-1}))$ of the ideal I of the ring S_r.

(2) The permutation π on the set of orthogonal idempotents e_i.

(3) The embedding α of the plaintext database D in the ring $S_n = S_r/I$.

5.2 One-to-one embeddings of $\mathbb{Z}_p$ in S_n

If elements of the original database do not have a ring structure (which is the case with, say, credit card numbers, phone numbers, etc.), then embedding of the original database in S_n does not have to be homomorphic, it can be just one-to-one. This is sufficient to be able to do private search on an encrypted database.

The number of one-to-one embeddings of $\mathbb{Z}_p$ in S_n is really big, even for a small number n of generators x_i, since S_n has p^{2^n} elements.

5.3 Fully homomorphic embeddings of $\mathbb{Z}_p$ in S_n

First we note that if g is an idempotent of a ring S, then for any ideal I of S, the coset $g+I$ is going to be an idempotent of the factor ring $R = S/I$.

Then, it is obvious that a fully homomorphic embedding α of $\mathbb{Z}_p$ in a ring S is completely determined by the image $\alpha(1)$ of the element 1. It is also obvious that $\alpha(1)$ should be an idempotent of S since 1 is an idempotent of $\mathbb{Z}_p$. The element 1 of $\mathbb{Z}_p$ can be mapped to any idempotent of S_n, including x_i, or $1-x_i$, or $x_i x_j,\ i \neq j$, etc. There are therefore 2^{2^n} fully homomorphic embeddings of $\mathbb{Z}_p$ in S_n, by the number of idempotents in S_n.

When we encrypt $\alpha(1)$ in S_r, the result is an idempotent of S_r plus "noise", i.e., plus an element of the ideal I of S_r. However, if S_r is published explicitly as a direct sum of copies of $\mathbb{Z}_p$ (i.e., if a basis of orthogonal idempotents of S_r is public), then any ideal of S_r just corresponds to a subset of direct summands. In other words, encryptions of 0 in the orthogonal basis have some of the coordinates random while other coordinates are just equal to 0, which is not good because encrypted nonzero elements may then be exposed, even though $1+E(0)$ may no longer be an idempotent of S_r.

To prevent this from happening, one can use some or all of the following enhancements. We particularly recommend to use (i) in implementations since it is the most practical one and the easiest one to implement, even though it may somewhat slow down computation on encrypted data. Our description below is not very formal, to better explain the idea.

(i) At Step 3 of the protocol in Section 5, we used a random permutation on the orthogonal basis as part of a secret encryption/decryption key. In view of what is said above, this may not be enough to diffuse $\alpha(k)$ in case of a fully homomorphic embedding of $\mathbb{Z}_p$. We therefore suggest to use a more general linear transformation of the orthogonal basis in this case. Specifically, after applying a random permutation on the orthogonal basis, we suggest to use a random "triangular" linear transformation of the form $f_j = \sum_{i \in F_j} e_{ij}$, where the sets F_j of indices have the property $F_k \subseteq F_j$ if $k \leq j$. This property implies that $f_j f_k = f_k$ if $k \leq j$. If this linear transformation is invertible, then f_j form a new basis of idempotents. They are no longer orthogonal, but the multiplication law $f_j f_k = f_k$ if $k \leq j$ makes multiplication in the public ring S_n still quite efficient.

To ensure that a triangular transformation is invertible, one can select it as follows. After applying a random permutation on the orthogonal basis of e_i, let $f_1 = e_1$. Then, for $j \geq 2,\ f_j = e_j + \sum_{i<j} c_i e_i$, where each coefficient c_i is randomly selected to be equal to either 0 or 1.

We note that the number of invertible triangular transformations as above is quite large. Indeed, at the last step alone, when choosing a linear combination for f_{2^r}, one has 2^{2^r-1} choices. With suggested parameter $r = 10$, this is a huge number.

The following two enhancements are not as practical as the one above, but we still include them as they might be of interest to algebraists.

(ii) At Step 2 of the protocol in Section 5, the ideal I (for encryptions of 0) can be generated not necessarily by $(x_m - w_m(x_1,\dots,x_{m-1}))$, but by $(x_{i_1}\dots x_{i_k}x_m - w_m(x_1,\dots,x_{m-1}))$, where all $i_j < m$, so that $x_{i_1}\dots x_{i_k}x_m$ is an arbitrary monomial involving x_m rather than just x_m itself. Since any monomial is an idempotent, this represents an isomorphic embedding of S_m into itself that is not *onto* unless the monomial is just x_m. Note that the monomial $x_{i_1}\dots x_{i_k}x_m$ will play the role of x_m in any further action, in the sense that whenever x_m appears in a further action, it should be in a monomial which is a multiple of the monomial $x_{i_1}\dots x_{i_k}x_m$. Then, during decryption, the monomial $x_{i_1}\dots x_{i_k}x_m$ is replaced by $w_m(x_1,\dots,x_{m-1})$, and this decryption is unique.

(iii) Recall that S_n has a canonical basis of 2^n orthogonal idempotents that we denote by e_i. Transition between the standard basis of x_i and the orthogonal basis of e_i is not very transparent, so in order to easier implement measures (such as (ii) above) to avoid exposure of nonzero elements of $\mathbb{Z}_p$ in ciphertext, one can do the following. Instead of choosing a generator of the ideal I in the form $(x_{i_1}\dots x_{i_k}x_m - w_m(x_1,\dots,x_{m-1}))$, one can choose it in the form $(x_{i_1}\dots x_{i_k}x_m - \sum_{i\in F} e'_i)$, where e'_i are elements of the orthogonal basis of S_{m-1} (converted to the standard basis) and F is a subset of the set of all indices. We note that each e'_i can be represented as a sum of monomials in $x_1,\dots,x_{m-1}$ with coefficients ("coordinates") 0 or ± 1. Then, in the standard basis of S_m, coordinates of e'_i will be $(e'_i, -e'_i)$, i.e., concatenation of coordinates of e'_i and $-e'_i$ in the standard basis of S_{m-1}.

Finally, we re-iterate that applying an enhanced procedure from this subsection may slow encryption down a bit, but as we have mentioned before, encryption is a one-time thing, so what really matters for real-life applications is how efficient multiplication in the public ring is. We claim that multiplication remains efficient due to the rule $f_j f_k = f_k$ if $k \le j$. With suggested parameters, the number of f_j in the basis is 2^{10}.

6 Ciphertext security

In this section, we explain why our encryption is secure against ciphertext-only attack. This is a very important security feature in typical real-life applications since it keeps private information secure in the event of a hackers' attack. We note that for a "regular" (non-homomorphic) private-key encryption ciphertext-only security is trivial to achieve. This is, however, not the case with FHE since in most cases, given several ciphertexts, an attacker can obtain several (plaintext, ciphertext) pairs using homomorphic properties of the encryption, and then use the known plaintext attack.

Note that with FHE, if original plaintexts come from $\mathbb{Z}_p$, then there are precisely three ways that one can possibly obtain any (plaintext, ciphertext) pairs based on ciphertexts only:

(1) Since the ring $\mathbb{Z}_p$ is commutative, for any two elements $a,b \in \mathbb{Z}_p$ we have $ab-ba=0$. By the homomorphic property of the encryption function E, we should therefore have $xy-yx=E(0)$ for any x,y in the encrypted database.

(2) For any $a \in \mathbb{Z}_p$, $a+a+\ldots+a=0$ (p times). Therefore, we should have $x+x+\ldots+x=E(0)$ (p times) for any x in the encrypted database.

(3) For any $a \in \mathbb{Z}_p$, $a \neq 0$, $a^{p-1}=1$ (by Fermat's little theorem). Therefore, we should have $x^{p-1}=E(1)$ for any x in the encrypted database.

In our FHE scheme, the ring S containing ciphertexts is isomorphic to the direct sum of 2^n copies of $\mathbb{Z}_p$, and therefore **(a)** $ab-ba=0$; **(b)** $a+a+\ldots+a=0$ (p times); **(c)** $a^{p-1}=1$ for any elements $a,b \in S$. Therefore, one cannot get any nontrivial (plaintext, ciphertext) pairs based on ciphertexts only. We note that the latter property (c) is the rare one and is not satisfied by most rings used for FHE in the literature.

7 Accumulating encryptions of zero and security of nonzero elements

Accumulating many encryptions of zero (or, more generally, of the same element) that use the same ideal I of the public ring is a security hazard in any FHE scheme based on rings, see e.g. [6]. Specifically, if an attacker accumulates a number of encryptions of zero exceeding the dimension of the ideal I (as a linear algebra over $\mathbb{Z}_p$), then he can recover this ideal by using linear algebra. Even though in our model we do not give access to the encryption mechanism to anybody except for the private key holder, encryptions of zero may be accumulated based on queries against the encrypted database and responses to these queries.

The most straightforward way of dealing with this issue is just to increase the dimension of the ideal I used for encryptions of zero. However, this will affect efficiency and ruin the hope to make FHE practical enough for performing operations on encrypted data (e.g. private search) that involve millions of multiplications.

A more practical way to protect the ideal I is to use a different ideal $J \supset I$ for encryptions of 0 every time a query against the encrypted database is made. This is described in our Section 8.1; in particular, see Remark 4.

In this section, we focus on a question of independent interest and importance: what can happen if the adversary figures out the ideal I? The dimension of the ideal I used for encryptions of zero in our scheme is rather small (about 1000 with suggested parameters), so accumulating about 1000 encryptions of zero will allow the adversary to determine the ideal I and recognize future encryptions of zero. However, we claim that this will not affect security of nonzero elements (which is what matters in most real-life applications) because to decrypt correctly, one has to know not just the ideal I itself, but also the specific map ρ (in the notation of our Section 2) that "kills" this ideal. We illustrate this point by a simple example from linear algebra.

Suppose a linear space S has a basis $\{x_1,x_2,x_3\}$, and a subspace I has a basis $\{x_3-x_1,x_2-x_1\}$. Then, if the map from S onto S/I is given by $x_1 \to x_1, x_2 \to x_1, x_3 \to x_1$, then, say, the vector $u=x_1+2x_2+3x_3$ will be taken to $6x_1$. At the same time, the subspace I has another basis: $\{x_3-x_1,x_3-x_2\}$. There is also another map from S onto

S/I such that $x_1 \to x_3, x_2 \to x_3, x_3 \to x_3$. Then the same vector u is taken to $6x_3$. There are, in fact, great many different maps from S onto S/I corresponding to different bases of I. For example, I has bases $\{x_3 - x_1, a(x_3 - x_1) + b(x_2 - x_1)\}$ for any nonzero a, b. The latter element of the basis is equal to $ax_3 + bx_2 - (a+b)x_1$. Thus, if $b \neq 0$, the following map takes S onto S/I: $x_1 \to x_1, x_2 \to \frac{a+b}{b}x_1 - \frac{a}{b}x_3,\ x_3 \to x_1$. Then the vector u is taken to $\frac{3b-a}{b}x_1$. Thus, in particular, any kx_1 can be "decryption" of u for some ρ.

Remark 1. The above argument shows that, if plaintexts do not have a ring structure (which is the case in most real-life applications) and therefore the embedding of plaintexts in S_n can be just one-to-one but not necessarily a full homomorphism, then essentially any $k \in \mathbb{Z}_p$ can be decryption of the same given $u \in S_r$.

Moreover, knowing an encryption of the element 1 of the ring $\mathbb{Z}_p$ (or any other element, for that matter) does not help the adversary to decrypt other elements correctly – again because the initial embedding $\alpha : \mathbb{Z}_p \to S_n$ is not a homomorphism, so that there is no algebraic correlation between encryption of, say, one credit card number and another. That said, since a one-to-one embedding of plaintexts in S_n is part of the private key, it has to have a description of a reasonable size, much smaller than the size of the plaintext database itself. We do not address this issue here, focusing on less trivial issues instead.

Remark 2. If plaintexts do have a ring structure, then the element 1 of the ring $\mathbb{Z}_p$ should be first mapped to an idempotent u of the ring S_n; this map then extends to a fully homomorphic embedding of $\mathbb{Z}_p$ into S_n by the homomorphism property. Then an encryption of the element $k \in \mathbb{Z}_p$ will be of the form $ku + e$, where $e \in I$. A computationally unbounded adversary who knows the ideal I can then use the following brute force attack on a ciphertext. Knowing that a ciphertext is of the form $ku + e$, the adversary can go over all elements $m \in \mathbb{Z}_p$ one at a time, divide the ciphertext $ku + e$ by m and check whether the result is an idempotent modulo the ideal I. If it is, then it must be of the form $v = u + e'$, where $e' \in I$. Knowing this and using the homomorphism property of the encryption, a computationally unbounded adversary can then recognize encryption of any element of $\mathbb{Z}_p$ as follows. Given a ciphertext w, the adversary can go over all elements $m \in \mathbb{Z}_p$ one at a time until she finds m such that $mv - w \in I$. Then w must be an encryption of this m.

8 Private search

We now recall how FHE is typically used for private search on an encrypted database; this functionality appears to be in high demand.

Suppose Alice (the data owner) wants to find out whether or not $E(x)$ is in the encrypted database $E(D)$. Recall that the encryption $E(x)$ has to be *randomized*, which means that every time Alice encrypts the same x the result looks different. This deprives the database keeper Carl (e.g. the cloud) from just matching $E(x)$ to elements of the encrypted database. Instead, Carl does the following. For each element $E(y)$ of the database, he computes $E(x) - E(y)$, which is equal to $E(x-y)$ modulo the ideal I that is used for encryptions of 0 (see our Section 2). Then he computes the product

$$P(x) = \Pi_{E(y) \in E(D)}(E(x) - E(y)) = \Pi_{E(y) \in E(D)} E(x-y)$$

over all elements of the database $E(D)$ and sends $P(x)$ to Alice. Since the encryption function E respects the multiplication, too, the element $P(x)$ is equal to $E(\Pi_{E(y)\in E(D)}(x-y))$ modulo the ideal I. Alice then decrypts this element to recover $\Pi_{E(y)\in E(D)}(x-y)$. If all plaintexts y are elements of a field (or more generally, of a ring without zero divisors), then the latter product is equal to 0 if and only if $x = y$ for at least one y such that $E(y)$ is stored in the database $E(D)$. Thus, Alice will know whether or not $E(x)$ is in the database $E(D)$, although this will not tell her how many occurrences of $E(x)$ there are in $E(D)$ (in case there are some).

The same method can be used if plaintexts are elements of a ring *with* zero divisors, provided that there are "not too many" of them, meaning that the probability to have $ab = 0$ for nonzero a, b is negligible. Thus, we are allowing a negligible probability of a "false positive" response to a query, while the probability of a "false negative" response is 0 since $ab \neq 0$ implies $a \neq 0$ and $b \neq 0$.

In our situation, the ring S_n is isomorphic to a direct sum of 2^n copies of $\mathbb{Z}_p$. Therefore, to have $\Pi u_i = 0$ for nonzero u_i, one should have a 0 (in one direct summand or another) in each of the 2^n components. Assuming that each element of $\mathbb{Z}_p$ in each component is (approximately) equally likely to occur, we have the probability of approximately $\frac{1}{p}$ for having 0 in a particular component. Thus, if we have a product of N elements u_i, the probability to have 0 in a particular component of the product Πu_i is bounded above by $\frac{N}{p}$. Then the probability to have 0 in *every* component of the product is bounded above by $(\frac{N}{p})^{2^n}$. Thus, if N is less than p, this probability tends to be quite small. With our tested parameters, p is on the order of 2^{30} and $n = 7$, so with N up to, say, $2^{27} \approx 10^8$, the probability of a "false positive" response is really small. Of course, the above argument is informal (in particular, it is not clear how close to uniform the distribution of elements of $\mathbb{Z}_p$ in each component is), but it gives a ballpark estimate of the probability of a "false positive" response. In fact, in our computer experiments with $N \approx 10^6$ multiplications, we never had a "false positive" response.

8.1 Third party private search

Despite the fact that our encryption is private-key and we do not give access to the encryption mechanism to any third party, we show below that a third party is still able to do private search on a privately encrypted database. This functionality is quite useful in scenarios similar to the following. Suppose one agency owns a list of names (e.g. a "no-fly list") that they do not want to share with anybody. Then another agency (or a company) wants to check whether a particular name is on this list, but they do not want to reveal the name.

Let Alice be the owner of the original (plaintext) database D, Carl the keeper of the encrypted (by Alice) database E(D), and Bob the third party who wants to do private search on E(D). In doing so, Bob does not want Alice (or anybody else) to know what he is looking for. At the same time, Alice does not want to share with Bob (or anybody else) her private encryption/decryption key. She would have to share though the way that real-life data are encoded by elements of the ring $R = S_n$. That is, she would have to share the map α in the notation of our Section 2.

The following protocol solves this problem. Recall that plaintexts are embedded in S_n, and the encrypted database $E(D)$ is a subset of $S_r,\ r > n$. We naturally identify S_n with the factor ring S_r/I, where the ideal I of S_r is used for encryptions of 0, see Section 5. If Alice wants to allow for a third party search on $E(D)$, she will have to publish $E(D)$ using the orthogonal basis of S_{r+k} for some $k \geq 1$ because the orthogonal basis of S_r does not extend to the orthogonal basis of S_{r+k}, and the third party will need the "extra" $k \geq 1$ generators x_i to do their private encryption.

Now suppose Bob encodes his real-life plaintext by an element $x \in S_n$ (the way to do it should be shared by Alice) and wants to find out whether an encryption of x is in the encrypted database $E(D)$, but he does not want to reveal x to Alice (or anybody else). Here is how he can do that.

1. Denote by $S' = S_{r+1}$ the ring generated by $x_1, \ldots, x_r, x_{r+1}$. Bob selects an element of the form $x_{r+1} - w_b(x_1, \ldots, x_n)$, where w_b is an idempotent, and let J be the ideal of S' generated (as an ideal of S') by this element.
2. Bob encrypts x as $E_B(x) = x + g$, where g is a random element of the ideal J of the ring S', as above. Bob then sends $E_B(x)$ to Alice. Note that $E_B(x)$ is expressed in terms of the "standard" generators x_i since Bob does not know how exactly the standard basis of S_{r+1} is to be converted to the orthogonal basis – this is part of Alice's private key.
3. Alice encrypts $E_B(x)$ as $E_A(E_B(x)) = E_B(x) + h$, where h is a random element of the ideal I of the ring S_r. The element $E_B(x) + h$ is expressed in terms of the public basic elements e_i of S_{r+1} before Alice sends it to Carl.
4. Carl (the encrypted database keeper) returns to Alice the product (see the beginning of Section 8)
   $P(x) = \Pi_{E(y) \in E(D)}(E_A(E_B(x)) - E_A(y)) = \Pi_{y \in D} E_A(E_B(x) - y) =$
   $= E_A(\Pi_{y \in D}(E_B(x) - y))$.
   These equalities are modulo the image of the ideal $I + J$ in the ring S_{r+1}.
5. Alice decrypts $P(x)$ with her decryption key and gets $Q(x) = \Pi_{y \in D}(E_B(x) - y)$ (modulo the ideal J), expressed in terms of the standard generators $x_1, \ldots, x_n, x_{r+1}$. She then sends $Q(x)$ to Bob.
6. Bob decrypts $Q(x)$ by replacing x_{r+1} with $w_b(x_1, \ldots, x_n)$ and gets $\Pi_{y \in D}(x - y)$. (See Remark 3 below for an explanation of why he actually gets this.) With overwhelming probability, the latter product is equal to 0 if and only if $x = y$ for at least one y.

Remark 3. Recall that $Q(x) = \Pi_{y \in D}(E_B(x) - y)$. This can be re-written as $\Pi_{y \in D}(x + g - y)$, since $E_B(x) = x + g$, where g is an element of the ideal J. Decrypting $Q(x)$ amounts to "killing" the ideal J by replacing x_{r+1} with $w_b(x_1, \ldots, x_r)$. This "kills" g but does not change x or y since these elements do not involve x_{r+1}. Therefore, decrypting $Q(x)$ indeed produces $\Pi_{y \in D}(x - y)$.

Remark 4. The same method can be used by the database owner to protect the ideal I from exposure during queries against the encrypted database. Every time a query is made, a different ideal J can be used to encrypt 0, thus hiding the "core" ideal I.

9 Performance

• With our suggested parameters, encryption of a single plaintext (i.e., of an element of S_7) takes 37 microseconds, hence encryption of a million plaintexts takes about 37 seconds. A single plaintext (with the suggested parameters) has size 3840 bits. This, however, is not so important as far as real-life applications are concerned since encryption is a "one-time thing". What matters is the speed of computation on encrypted data, see the next bullet point.

• We do one million multiplications of encrypted elements (in the orthogonal basis of S_{10}) in 2 sec on a regular laptop computer, without any optimization or parallelization. A recent improvement on Gentry's algorithm [4] (tacitly) implies a single multiplication in about 0.7 sec, hence a million multiplications in about 700,000 seconds, which means we do it roughly 350,000 times faster.

• Private search on an encrypted database with a million entries takes 3.6 sec. on a regular laptop computer, again without any optimization or parallelization.

10 Conclusions

• We described a practical private-key fully homomorphic encryption scheme based on rings and showed that it provides for efficient computation on encrypted data.

• Even though our encryption is private-key and we do not give access to the encryption mechanism to any third party, a third party is still able to do private search on a privately encrypted database.

• One of the main distinctive features of our scheme is the presence of an orthogonal basis of idempotent elements in the public ring. The fact that it is orthogonal makes multiplication of encrypted elements (the main stumbling point in other schemes) extremely efficient.

• Our encryption scheme is unconditionally secure against ciphertext-only attack.

• Typically, security of FHE schemes rests on masking encryptions of 0 by "noise". In any scheme, accumulating sufficiently many encryptions of 0 leads to the ability to recognize other encryptions of 0. What happens in most ring-based FHE schemes is that the speed of multiplication of encrypted elements grows linearly with the dimension of the ideal used for encryptions of 0. This is also the case with our scheme. However, in contrast with other schemes, recovering the ideal used for encryptions of 0 in our scheme *does not recover the decryption key*, and nonzero elements remain secure.

References

1. Z. Brakerski, C. Gentry and V. Vaikuntanathan, *(Leveled) fully homomorphic encryption without bootstrapping*, in: ITCS 2012 – Proceedings of the 3rd Innovations in Theoretical Computer Science Conference, pp. 309–325.
2. Z. Brakerski and V. Vaikuntanathan, *Efficient fully homomorphic encryption from (standard) LWE*. In: FOCS 2011, 97-106, IEEE Computer Soc., Los Alamitos, CA, 2011.
3. M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, *Fully homomorphic encryption over the integers*. In: Advances in Cryptology – EUROCRYPT 2010, Lecture Notes Comput. Sci. **6110** (2010), 24-43.

4. L. Ducas and D. Micciancio, *FHE Bootstrapping in less than a second*, in: Advances in Cryptology – EUROCRYPT 2015, Lecture Notes Comput. Sci. **9056**, 617–640.
5. C. Gentry, *Fully homomorphic encryption using ideal lattices.* In: STOC 09: Proceedings of the 41st annual ACM Symposium on Theory of Computing, pp. 169-178. ACM, New York, NY, USA (2009).
6. D. Grigoriev, I. Ponomarenko, *Homomorphic public-key cryptosystems over groups and rings*, Quaderni di Matematica **13** (2004), 305–326.
7. S. Halevi and V. Shoup, *HElib - An Implementation of homomorphic encryption.* https://github.com/shaih/HElib/
8. S. Halevi and V. Shoup, *Algorithms in HElib*, in: CRYPTO 2014, Lecture Notes Computer Sci. **8616**, 554–571.