A note on fully homomorphic encryption of real-life data

Delaram Kahrobaei[1] and Vladimir Shpilrain[2]

[1] City University of New York
dkahrobaei@gc.cuny.edu
[2] City University of New York
vshpilrain@ccny.cuny.edu

Abstract. The purpose of this short paper is to explain the difference between encrypting real-life data and encrypting elements with a ring structure in the context of fully homomorphic encryption (FHE). Specifically, our encryption of real-life data is in two stages. First, we use a private-key embedding of real-life data in a ring; this embedding does not have to be fully homomorphic. This private embedding, speaking informally, takes most of the security burden off the second part of our encryption procedure, namely FHE between rings. The whole encryption function is then not fully homomorphic, but we show that it still provides for the most popular functionalities one expects from FHE, including private search on encrypted data.

1 Introduction

In [7], we offered a practical fully homomorphic encryption (FHE) method where computation on encrypted data was several orders of magnitude faster than with established (by that time) FHE schemes, including [1], [2], [3], [4], [5], [10]. This raised a natural suspicion about security of our FHE scheme. In the present paper, we explain the difference between the above work of theoretical cryptographers and our paper [7]. For security analysis, literature survey, performance, etc. we refer to [7]; these aspects will not be reproduced here. We do reproduce our algorithm though, to make the explanations easier to follow.

In all of the above cited works by theoretical cryptographers, the main obstacle to efficient computation on encrypted data was the need to protect encryptions of 0 (if we talk about encrypting elements of a ring). This obstacle is essentially insurmountable if one wants to have a practical FHE, as we will try to explain below.

Let $E : R \to S$ be a fully homomorphic encryption from a ring R *onto* a ring S, i.e., every element of S can be an encryption of an element of R. Then encryptions of 0 form an ideal J of the ring S because of the homomorphic properties. (Elements of J are often called "noise".) The homomorphic properties then are:

$$E(x+y) = E(x) + E(y) \pmod{J}$$

$$E(xy) = E(x)E(y) \pmod{J}.$$

During the decryption process, elements of the ideal J are mapped to $0 \in R$, so the composition of encryption and decryption works exactly as we want it to, as far as homomorphic properties are concerned.

Now note that it is easy for an adversary who knows just one encryption of 0 to obtain many encryptions of 0. This is because if $u_1, u_2 \in S$ are encryptions of 0, then for any $s_1, s_2 \in S$, the element $u_1 \cdot s_1 + u_2 s_2$ is an encryption of 0, too, by the homomorphic properties. Therefore, upon getting a hold of an encryption of 0, the adversary Eve can basically get as many of them as she wants. Having done that, Eve can "compute" the ideal J. What this means is that Eve can find a basis of J as a linear subspace of S. With such a basis of J in hand, Eve can then distinguish encryptions of 0 from encryptions of other elements. This means encryption is not semantically secure, and this is not acceptable to theoretical cryptographers.

The only way to prevent this from happening is to make the dimension of the ideal J huge, so that computing a basis of J would be computationally infeasible. However, then there is a problem with efficiency when one does computations (specifically, multiplications) on encrypted data, and here is why. Let $0 \neq w \in R$. For semantic security, $E(w)$, the encryption of w, should be randomized, and in particular have many possible values. Suppose $u \in S$ and $v \in S$ are both encryptions of w. Then $u - v$ must be an encryption of 0, by the homomorphic property. This implies that all encryptions of w are of the form $x_w + e$, where $x_w \in S$ is a particular encryption of w and $e \in J$ an encryption of 0. Now, if the dimension of the ideal J is huge, then when you multiply out elements of the form $x_w + e$, the number of different "monomials" involving various $e \in J$ keeps growing exponentially. The only visible solution to this problem is to occasionally remove these monomials during computation because they are going to go to 0 during the decryption process anyway. This is what is called "bootstrapping", and all the efforts since [5] have been focused on improving the bootstrapping process; this includes, in particular, search for a platform ring where bootstrapping would be most efficient.

These efforts had some success, but improving efficiency to the point where, say, a private search on an encrypted database with millions of entries would be possible in reasonable time, is still out of reach.

1.1 How our approach is different

In [7], we took a different approach, focusing on real-life data and real-life applications of FHE from the beginning. Having figured out that combining efficiency and semantic security in FHE is essentially impossible (this was actually pointed out earlier in [8]), we have sacrificed semantic security to some extent by giving up security of encryptions of 0. We realize that this will make theoretical cryptographers run for exit, but the whole idea of FHE, since inception, was to be applied to specific real-life problems, so let us look at encryptions of 0 from the real-life perspective.

Suppose the adversary Eve learns how to recognize encryptions of 0. If she cannot recognize encryptions of any other number, this does not seem to give her access to any protected data since real-life data (like credit card numbers, phone numbers, medical data, etc.) are typically represented by nonzero numbers. The only information Eve is able to obtain is, during a private search on encrypted database, whether or not the search has resulted in a match. Of course, in some rare situations this information may be valuable, but then again – in most cases, if a search has resulted in a match, it is followed by retrieving some information, so the adversary would know if there was a

match anyway. Thus, compromising encryptions of 0 in most real-life situations does not give any valuable information to the adversary.

Our encryption scheme works like this:

$$D \xrightarrow{\tau} R \xrightarrow{E} S.$$

Here D is a set of private real-life data, τ is a private map, R and S are rings, and E is a private-key fully homomorphic encryption. The composite encryption $E(\tau)$ is therefore not fully homomorphic, but we will show in Section 3 that our scheme still allows for a private search on encrypted data. We also note that in our scheme in [7], the ring R was a *retract* of S, i.e., it was a subring of S and at the same time isomorphic to a factor ring of S. We give more details in Section 2; here we just mention that the ring S has to be public since this is where all computations on encrypted data take place.

The encryption $E(u)$ of an element $u \in R$ can be just $s+e$ for a randomized $e \in J$. For particular platform rings R, S used in our paper [7], we briefly discuss encryption in Section 2.

We also note that the private map τ should be one-to-one, i.e., an embedding. Some functionalities, including statistical analysis and machine learning on encrypted data, may require τ to also be monotone, i.e., one should have $\tau(d_1) < \tau(d_2)$ whenever $d_1 < d_2$ for $d_1, d_2 \in D$. In that case, the order of elements should be somehow defined on the ring S. See our papers [6], [11], [12] for details on how this method can be applied to analysis of encrypted medical data.

2 Platform ring

The ring that we used as the ring $R = R_n$ in our paper [7] was a direct sum of 2^n copies of the ring $\mathbb{Z}_p$, for a small n, e.g. $n = 7$ or $n = 8$. The ring S is isomorphic to R_{n+1}, and the isomorphism is private; we discuss isomorphisms below.

We note that by the homomorphic properties, encryption of any element of R is determined by encryptions of special "quasi-unit" elements q_i. These are vectors of dimension 2^n (over $\mathbb{Z}_p$) that have $1 \in \mathbb{Z}_p$ in the ith coordinate and 0 in all others. The sum of all different "quasi-unit" elements q_i is the actual unit element $\bar{1}$ of S, i.e., the vector of dimension 2^n all of whose coordinates are equal to $1 \in \mathbb{Z}_p$.

Since any "quasi-unit" element $q = q_i$ is *idempotent*, i.e., $q_i^2 = q_i$, it should be encrypted as $E(q) = v + e$, where $e \in J$ and v is an idempotent element of S. What are the idempotent elements of the ring S? They are vectors of dimension 2^{n+1} where some of the coordinates are equal to 1 and the others are 0. Therefore, the number of idempotents equals the number of subsets of a set with 2^{n+1} elements, i.e., equals $2^{2^{n+1}}$. Thus, even with a small n (we suggested $n = 7$ or $n = 8$) the number of idempotents is huge.

Similar considerations apply to *isomorphisms* of S. Specifically, for any subset I of the set $\{1, \dots, n+1\}$, let q_I be the vector of dimension 2^{n+1} (over $\mathbb{Z}_p$) that has $1 \in \mathbb{Z}_p$ in the coordinates indexed by the elements of I and 0 elsewhere. Note that for any I, q_I is an idempotent. Any collection of $(n+1)$ linearly independent vectors $x_k = q_{I_k}$, $k = 1, \dots, n+1$, determines an additive homomorphism of S onto itself. Multiplication rules, however, can change: for the elements q_i, one has $q_i q_j = 0$ if $i \neq j$, but in general,

one has $x_j x_k = \sum_t x_t$ for some x_t. Somewhat surprisingly, these multiplication rules do not have to be given to the public, as we explain in Section 2.2.

As for the number of different isomorphisms of S, we note that any permutation on the set $\{q_1, \ldots, q_{2^{n+1}}\}$ determines an isomorphism of S. Thus, the number of different isomorphisms of S is at least $(2^{n+1})!$, which is already a huge number.

2.1 Ideal J of encryptions of 0

Ideals of $S = R_{n+1}$ are quite easy to describe. Any ideal consists of vectors of dimension 2^{n+1} whose coordinates at specific places are 0. Therefore, the number of different ideals of S is $2^{2^{n+1}}$, but the maximum dimension of an ideal is 2^{n+1}, i.e., is rather small given that suggested values for n in [7] were $n = 7$ or $n = 8$. This is why Eve can determine the ideal J upon accumulating a few hundred different encryptions of 0.

2.2 Multiplication and decryption

Recall that in our paper [7], the ring $R = R_n$ was a direct sum of 2^n copies of the ring $\mathbb{Z}_p$, and the ring S was isomorphic to R_{n+1}, i.e., it was a direct sum of 2^{n+1} copies of $\mathbb{Z}_p$.

Then, in [7] we interpreted the (public) ring S as a ring of polynomials over $\mathbb{Z}_p$ in $(n+1)$ commuting idempotent variables $x_1, \ldots, x_{n+1}$ (i.e., $x_i^2 = x_i$). Note that monomials in these variables can have degree at most $(n+1)$ because of the idempotency identities $x_i^2 = x_i$. This implies that there are at most 2^{n+1} different monomials, and this makes computation with these polynomials very efficient.

The "quasi-unit" elements (one can also call them "coordinate vectors") q_j (see Section 2) can be expressed as products of the form $\Pi(1 - x_{i_k})$, for various subsets of x_i. In turn, each x_i can be expressed as a linear combination of q_j; in fact, each x_i is equal to some q_I in the notation of Section 2. In other words, each x_i is just a sum of some q_j. (Note that $q_i q_j = 0$ if $i \neq j$.)

Therefore, a (private) linear transformation of coordinate vectors q_j entails a (private) linear transformation of x_i, and this provides a good and efficient mechanism for hiding coordinates of encrypted nonzero elements. One has to be careful though that new variables x_i' are again idempotent; otherwise computation with polynomials in these variables will not be efficient. The new variables x_i' will be idempotent if and only if they are sums of some q_j. This narrows down the collection of linear transformations that can be used in this context, but it is still super-exponential in n. We refer to [7] for more details.

Now recall our encryption scheme:

$$D \xrightarrow{\tau} R \xrightarrow{E} S.$$

One part of the decryption process should be applying the inverse of the linear transformation mentioned above, to get to the "standard" variables x_i. Another part is "killing" the ideal J of encryptions of 0. These two parts of the decryption process can be applied in either order, depending on whether Alice has defined the ideal J in terms of the "standard" idempotent variables x_i or the new ones, x_i'. In our paper [7],

the ideal J was defined by specific generating polynomials in the "standard" variables, and "killing" this ideal was conveniently described as replacing a particular variable by a particular polynomial in other variables. In that case, applying the inverse linear transformation should be done first, followed by "killing" the ideal J.

There are several other ways of interpreting the ring S and describing the decryption procedure, but the bottom line is: without knowing the private decryption key, it is impossible to decrypt without ambiguity.

The last stage of decryption is just applying τ^{-1}, which is, again, private.

3 Private search

In this section, we show how the database owner, call her Alice, can run a private search on an encrypted database even if the composite encryption function is not fully homomorphic. The reason is simple: since the first, non-homomorphic part τ of our encryption function is one-to-one, Alice, who wants to search for encryption of $d \in D$, can run the search for $E(\tau(d))$ using the FHE part of the encryption, and then, whatever the result of the search for encryption of $\tau(d)$ is, it will carry over to d.

We now recall how FHE is typically used for private search on an encrypted database; this functionality appears to be in high demand. Recall our encryption scheme once again:

$$D \xrightarrow{\tau} R \xrightarrow{E} S.$$

Denote $\tau(D)$ by T. Suppose $x \in R$ and Alice wants to find out whether or not $E(x)$ is in $E(T)$. Recall that the encryption $E(x)$ has to be *randomized*, which means that every time Alice encrypts the same x the result may look different. This deprives the database keeper Carl (e.g. the cloud) from just matching $E(x)$ to elements of $E(T)$. Instead, Carl does the following. For each element $E(y)$ of $E(T)$, he computes $E(x)-E(y)$, which is equal to $E(x-y)$ modulo the ideal J that is used for encryptions of 0 (see our Section 2.1) since the encryption function E respects the addition.

Then he computes the product

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

over all elements of the database $E(T)$ and sends $P(x)$ to Alice. Since the encryption function E respects the multiplication, too, the element $P(x)$ is equal to

$$E(\Pi_{E(y)\in E(T)}(x-y))$$

modulo the ideal J. Alice then decrypts this element to recover $\Pi_{E(y)\in E(T)}(x-y)$. If all plaintexts $y \in T$ 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 $E(T)$. Thus, Alice will know whether or not $E(x)$ is in the database $E(T)$, although this will not tell her how many occurrences of $E(x)$ there are in $E(T)$ (in case there are some).

The same method can be used if plaintexts $y \in T$ 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 is isomorphic to a direct sum of M copies of $\mathbb{Z}_p$, where M is on the order of several hundred. 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 M coordinates. Assuming that each element of $\mathbb{Z}_p$ in each coordinate is (approximately) equally likely to occur, we have the probability of approximately $\frac{1}{p}$ for having 0 in a particular coordinate. Thus, if we have a product of N elements u_i, the probability to have 0 in a particular coordinate of the product Πu_i is bounded above by $\frac{N}{p}$. Then the probability to have 0 in *every* coordinate of the product is bounded above by $(\frac{N}{p})^M$. 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 $M = 128$, so with N up to, say, $10^8 \approx 2^{27}$, the probability of a "false positive" response is really negligible. 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.

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. A. Gribov, K. Horan, J. Gryak, D. Kahrobaei, R. Soroushmehr, V. Shpilrain, K. Najarian, *Medical diagnostics based on encrypted medical data*, in: Bio-inspired Information and Communications Technologies (BICT 2019), Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering **289** (2019), 98–111.
7. A. Gribov, D. Kahrobaei, V. Shpilrain, *Practical private-key fully homomorphic encryption in rings*, Groups, Complexity, and Cryptology **10** (2018), 17–27.
8. D. Grigoriev, I. Ponomarenko, *Homomorphic public-key cryptosystems over groups and rings*, Quaderni di Matematica **13** (2004), 305–326.
9. S. Halevi and V. Shoup, *HElib - An Implementation of homomorphic encryption.* https://github.com/shaih/HElib/
10. S. Halevi and V. Shoup, *Algorithms in HElib*, in: CRYPTO 2014, Lecture Notes Computer Sci. **8616**, 554–571.

11. A. Wood, D. Kahrobaei, V. Shpilrain, K. Najarian, *Private naive Bayes classification of personal biomedical data: application in cancer data analysis*, Computers in Biology and Medicine **105** (2019), 144–150.
12. A. Wood, K. Najarian, D. Kahrobaei, *Homomorphic Encryption for Machine Learning in Medicine and Bioinformatics*, ACM Computing Surveys **53** (2020).