(Finite) Field Work: Choosing the Best Encoding of Numbers for FHE Computation

Angela Jäschke(✉) and Frederik Armknecht

University of Mannheim, Mannheim, Germany
{jaeschke,armknecht}@uni-mannheim.de

Abstract. Fully Homomorphic Encryption (FHE) schemes operate over finite fields while many use cases call for real numbers, requiring appropriate encoding of the data into the scheme's plaintext space. However, the choice of encoding can tremendously impact the computational effort on the encrypted data. In this work, we investigate this question for applications that operate over integers and rational numbers using p-adic encoding and the extensions p's Complement and Sign-Magnitude, based on three natural metrics: the number of finite field additions, multiplications, and multiplicative depth. Our results are partly constructive and partly negative: For the first two metrics, an optimal choice exists and we state it explicitly. However, for multiplicative depth the optimum does not exist globally, but we do show how to choose this best encoding depending on the use-case.

Keywords: Fully Homomorphic Encryption · Encoding · Efficiency

1 Introduction

Fully Homomorphic Encryption (FHE) schemes allow arbitrary computations on encrypted data. Though many works have focused on improving the efficiency of FHE schemes themselves, an often overlooked aspect that strongly impacts performance is *how* an FHE scheme is applied. For instance, most FHE schemes operate over finite fields $GF(p^k)$ for an arbitrary prime p and $k \geq 1$, while many use cases call for natural, integer, rational or even real numbers, requiring appropriate encoding of the data into the scheme's plaintext space.

Thus, a naturally arising question is ***how to best encode the plaintext data so that later, operations on the encrypted data incur an overhead as small as possible.*** In this work, we analyze the effort for FHE computation subject to different p-adic encoding choices like the size of p and the embedding into $\mathbb{Q}$. We base our analysis on the following three natural cost metrics that arise when embedding the plaintext data into the $GF(p^k)$ structure:

Multiplicative Depth: All current FHE schemes are noise-based with each multiplication doubling the amount of noise, and when a noise threshold is

https://doi.org/10.1007/978-3-030-02641-7_23

passed, ciphertexts cannot be decrypted correctly anymore. *Bootstrapping* can remove some of the noise before it exceeds this threshold, but this is a very costly operation. Thus, the goal is often to minimize the number of bootstrappings that are necessary by minimizing the number of consecutive multiplications, also referred to as *multiplicative depth*. For this reason, multiplicative depth has been the standard cost metric and is naturally part of our analysis.

Number of Field Multiplications: FHE multiplications are much more expensive than additions for all current schemes, so keeping track of this number is an obvious choice. In addition, the multiplicative depth in p-adic encoding quickly becomes so large that bootstrapping is unavoidable, so that minimizing the total number of multiplications can speed up performance significantly.

Number of Field Additions: For all schemes today, field additions cost almost nothing compared to field multiplications. However, there is no theoretical reason why this must be the case, so we include this metric because it might be valuable in the future for a different kind of scheme.
Our contributions are:

- We derive a generic formula that allows to express each digit when adding two numbers in p^k-adic encoding[1].
- Based on the generic formula, we analyze the costs for adding two encrypted natural numbers in p^k-adic encoding with the following results:
  1. For the required number of field additions or field multiplications when $k = 1$, the efforts for additions and multiplications of encrypted integers strictly increase with p, making $p = 2$ by far the best choice.
  2. For the depth metric when $k = 1$, the optimal p depends heavily on the use case and our formulas show how to compute it.
  3. For p^k-adic encoding with $k > 1$, we show that performance is always worse compared to p-adic encoding, so setting $k = 1$ is the best choice.
- We then extended our analysis to negative and rational numbers.

2 Related Work

This section presents only the most relevant publications, a more comprehensive version can be found in the extended paper [10]. The recent increase in papers regarding encoding for FHE shows its importance: [5] encodes rational numbers through continued fractions (only positive rationals and evaluating linear multivariate polynomials), whereas [6] focuses on efficiently embedding the computation into a single large plaintext space. A work that explores similar ideas as [6] and offers an implementation is [8]. [1] allows floating point numbers, and [3] gives a high-level overview of arithmetic methods for FHE, but restricted to positive numbers. In [9], arithmetic operations and different binary encodings for

[1] The term p^k-adic encoding denotes the natural extension of p-adic encoding to the field $GF(p^k)$ for $k \geq 1$ and is explained in Sect. 6.

rational numbers are examined and compared in their effort. [2] explores a non-integral base encoding, and [13] presents different arithmetic algorithms including a costly division, though apparently limited to positive numbers. Lastly, [4] allows approximate operations by utilizing noise from the encryption itself. To our knowledge, there are no papers concerned with the costs of encoding in a base other than $p = 2$ except [11], which exclusively analyzes [12] and uses different cost metrics. The latter also presents a formula for the carry of a half adder, but merely considers $GF(p^k)$ for $k = 1$ in the context of homomorphically computing the decryption step (needed for bootstrapping) of their variation of [7], and does not include an effort analysis.

3 Formula for Computing Carry Values over $\mathbb{Z}_p$

In this section and the following Sect. 4, we lay the foundation for the effort analysis starting from Sect. 5. We derive in this section the formulas for the digits of the sum of two numbers in p-adic encoding. We will see that the carry is particularly important, so we investigate it more closely in Sect. 4.

Let $A = a_n a_{n-1} \ldots a_1 a_0$ and $B = b_n b_{n-1} \ldots b_1 b_0$ be two p-adically encoded natural numbers. If we wish to add these numbers in this encoding, we can write

$$\begin{array}{cccccccc} & & a_n & a_{n-1} & \ldots & a_2 & a_1 & a_0 \\ + & & b_n & b_{n-1} & \ldots & b_2 & b_1 & b_0 \\ \hline = & c_{n+1} & c_n & c_{n-1} & \ldots & c_2 & c_1 & c_0 \end{array} \tag{1}$$

To homomorphically evaluate a function on encrypted data, we need to express the result as a polynomial in the inputs – we need to be able to write

$$c_i = c_i(a_n, b_n, a_{n-1}, b_{n-1}, ..., a_1, b_1, a_0, b_0) \tag{2}$$

for any i, where $c_i(\ldots)$ refers to some polynomial. Clearly, it holds that $c_i = a_i + b_i + r_i$, where $r_0 = 0$, and for $i > 0$, r_i is the *carry* from position $i - 1$. Our goal in this section is to express $r_i(a_{i-1}, b_{i-1}, r_{i-1})$ as a polynomial, which will constitute Theorem 1. Addition is defined mod p, and we will often write r_i instead of $r_i(a_{i-1}, b_{i-1}, r_{i-1})$ for simplicity.

Theorem 1. *The formula for computing the carry $r_i(a_{i-1}, b_{i-1}, r_{i-1})$ is*

$$r_i(a,b,r) = \sum_{k=1}^{p-1} \Big(l_k(b) \cdot \sum_{j=1}^{k} l_{p-j}(a) \Big) + r \cdot (p-1) \cdot l_{p-1}(a+b) := f_1(a,b) + r \cdot f_2(a,b)$$

where $l_i(x) = \prod_{j=0, j \neq i}^{p-1} (x - j)$. This polynomial is unique in that there is no other polynomial of smaller or equal degree which also takes on the correct values for r_i at all points $(a_{i-1}, b_{i-1}, r_{i-1})$ with $a_{i-1}, b_{i-1} \in \{0, \ldots, p-1\}, r_{i-1} \in \{0, 1\}$.

The proof is given in the extended version of this paper [10].

4 The Effort of Computing the Carry

In this section, we present the effort required to compute each digit c_i when adding two natural numbers encoded p-adically. Due to space constraints, the detailed derivations of these numbers can be found in the extended version of this paper [10]. Recall that $c_i = a_i + b_i + r_i$, and r_i can be computed as $r_i = f_1(a_{i-1}, b_{i-1}) + r_{i-1} \cdot f_2(a_{i-1}, b_{i-1})$. Note that there cannot be any cancellation between the terms of $f_1(a_{i-1}, b_{i-1})$ and $r_{i-1} \cdot f_2(a_{i-1}, b_{i-1})$ due to the variable r_{i-1}. Thus, to compute c_i, we must compute f_1 and f_2, and additionally perform 3 field additions and 1 multiplication. Regarding depth, note that $r_0 = 0$ and $r_1 = f_1(a_0, b_0)$ (thus having the depth of f_1), and subsequent r_i have a depth of $\max\{\mathtt{D}(f_1), \max\{\mathtt{D}(r_{i-1}), \mathtt{D}(f_2)\} + 1\}$. Using this formula for r_2, we get

$$\mathtt{D}(r_2) = \max\{\lceil \log_2(p) \rceil, \lceil \log_2(p-1) \rceil\} + 1 = \lceil \log_2(p) \rceil + 1$$

From here on, it is clear that $\mathtt{D}(r_i) > \mathtt{D}(f_1) \geq \mathtt{D}(f_2)$, so the depth will increase by 1 with each i, leaving us with a total depth of $\mathtt{D}(r_i) = \lceil \log_2(p) \rceil + i - 1$. The effort required to compute each digit c_i can be found in Table 1.

Table 1. Effort for computing c_i.

Effort	Field additions	Field multiplications	Depth
f_1	$4p - 6$	$2p \cdot \log_2(p) + p - 2 \cdot \log_2(p) - 3$	$\lceil \log_2(p) \rceil$
f_2	$p - 1$	$p - 2$	$\lceil \log_2(p-1) \rceil$
Additional	3	1	+1
Total $\boldsymbol{c_i}$	$\mathbf{5p - 4}$	$\mathbf{2p \log_2(p) + 2p - 2 \log_2(p) - 4}$	$\mathbf{\lceil \log_2(p) \rceil + i - 1}$

Special Cases: The effort for $c_0 = a_0 + b_0$ is only 1 field addition, and that for $c_1 = a_1 + b_1 + r_1 = a_1 + b_1 + f_1(a_0), b_0)$ is $4p + 4$ additions, $2p \cdot \log_2(p) + p - 2 \cdot \log_2(p) + 1$ multiplications, and $\lceil \log_2(p) \rceil$ depth. Another special case is $c_{n+1} = r_{n+1}$, which has 2 field additions less than the other $c_i, i > 1$.

5 Cost Analysis for Encrypted Natural Numbers

5.1 The Cost of Adding Two Natural Numbers

Suppose we have some natural number $x > p$ (in decimal representation). Thus, x will be encoded with $2 \leq \ell := \lfloor \log_p(x) \rfloor + 1$ digits p-adically and the result will have $\ell + 1$ digits. We have $c_0 = a_0 + b_0$ with an effort of 1 addition. Next, we have $c_1 = a_1 + b_1 + r_1 = a_1 + b_1 + f_1(a_0, b_0)$ with an effort of $4p - 4$ additions, $2p \cdot \log_2(p) + p - 2 \cdot \log_2(p) - 3$ multiplications and a depth of $\lceil \log_2(p) \rceil$. The last digit $c_\ell = r_\ell$ has a cost of $5p - 6$ additions, $2p \cdot \log_2(p) + 2p - 2 \cdot \log_2(p) - 4$ multiplications, and a depth of $\lceil \log_2(p) \rceil + 1$. The remaining $\ell - 2$ middle digits c_i have the normal effort of $5p - 4$ additions, $2p \cdot \log_2(p) + 2p - 2 \cdot \log_2(p) - 4$ multiplications and a depth of $\lceil \log_2(p) \rceil + i - 1$.

In total, the cost of adding two ℓ-digit numbers, $\ell > 2$, is:

- $9p - 9 + (\ell - 2) \cdot (5p - 4) = (5\ell - 1) \cdot p - (3\ell + 2)$ **field additions**
- $4p \cdot \log_2(p) + 3p - 4 \cdot \log_2(p) - 7 + (\ell - 2) \cdot (2p \cdot \log_2(p) + 2p - 2 \cdot \log_2(p) - 4)$ $= 2\ell \cdot p \cdot \log_2(p) + (2\ell - 1) \cdot p - 2\ell \cdot \log_2(p) - 4 \cdot \ell + 1$ **field multiplications**
- **A multiplicative depth of** $\lceil \log_2(p) \rceil + \ell - 1$**.**

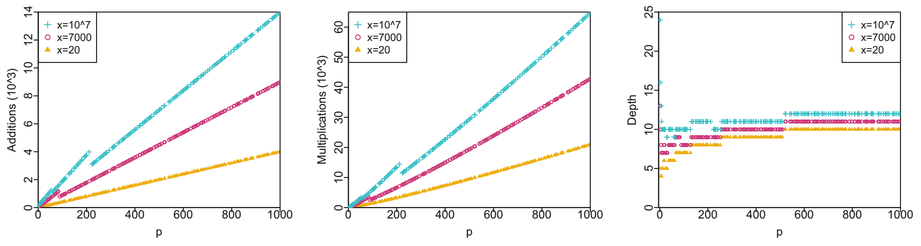
Note that for $0 \leq x \leq p - 1$, we only require one digit, which incurs a lower effort: **In total, the cost of adding two 1-digit numbers is** $4p-5$ **additions,** $2p \cdot \log_2(p) + p - 2 \cdot \log_2(p) - 3$ **multiplications and a depth of** $\lceil \log_2(p) \rceil$**.**

Recalling $\ell := \lfloor \log_p(x) \rfloor + 1$, we can now state the main result of this paper:

Theorem 2. *Using total number of additions or multiplications (or a balance between total number of multiplications and depth) as the cost metric, $p = 2$ is the most efficient encoding for adding two natural numbers in p-adic encoding.*

Proof. We can see that while the required encoding length $\ell = \lfloor \log_p(x) \rfloor + 1 = \lfloor \frac{\log_2(x)}{\log_2(p)} \rfloor + 1$ only decreases logarithmically, the effort grows with p as $\mathcal{O}(\ell \cdot p) = \mathcal{O}((\lfloor \frac{\log_2(x)}{\log_2(p)} \rfloor + 1) \cdot p) \approx \mathcal{O}(p + \frac{p}{\log_2(p)})$ (for additions) and as $\mathcal{O}(\ell \cdot p \cdot \log_2(p)) = \mathcal{O}((\lfloor \frac{\log_2(x)}{\log_2(p)} \rfloor + 1) \cdot p \cdot \log_2(p)) \approx \mathcal{O}(p \cdot \log_2(p) + p)$ (for multiplications). The depth $\lceil \log_2(p) \rceil + \ell - 1 = \lceil \log_2(p) \rceil + \lfloor \frac{\log_2(x)}{\log_2(p)} \rfloor$ also increases logarithmically.

We again point out that if the function being evaluated is known beforehand, choosing p so large that computations do not wrap around mod p is likely to be faster – however, this is not *Fully* Homomorphic Encryption but rather *Somewhat* Homomorphic Encryption. Theorem 2 holds for p-adic encoding with true FHE.



(a) Additions (b) Multiplications (c) Depth

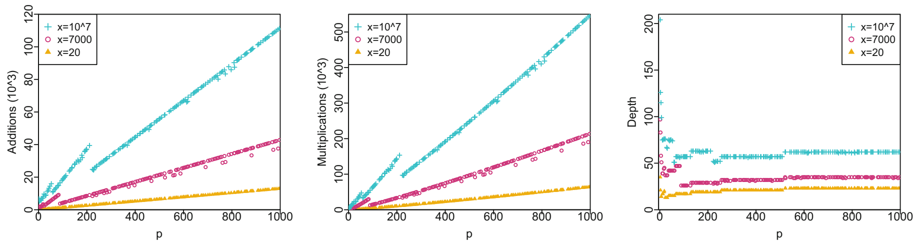
Fig. 1. Number of field additions, multiplications and depth for adding $x = 20/7000/10^7$ to a number of same size. The x-axis is the encoding base p, the y-axis is number of operations/depth, and the plots correspond to the three numbers.

We illustrate this Theorem through Fig. 1, which shows the effort as p grows for selected values of x. We see that the number of additions, multiplications and the depth increase significantly as the encoding base p increases. Note that the jags in the first two diagrams occur when the base prime becomes so large that one digit less is required for encoding than under the previous prime, so the effort drops briefly before increasing again. The diagram for depth shows us an interesting phenomenon that is hidden in the asymptotic analysis: For low primes, the required number of digits dominates the total depth cost. This problem becomes more pronounced the larger the encoded number is, and vanishes after the first few primes as the expected asymptotic cost takes over. This means that if depth is the only cost metric that is being considered, choosing a slightly larger prime than 2 yields better results at the cost of significantly increased multiplications. Also, the optimal choice of p depends heavily on the numbers that are being encoded. For example, in Fig. 1c, the depth-optimal choices for adding x would be $p = 3$ for $x = 20$, $p = 7$ for $x = 7000$, and $p = 29$ for $x = 10^7$.

5.2 The Cost of Multiplying Two Natural Numbers

We now analyze the cost of multiplying two natural numbers in p-adic encoding with the standard multiplication algorithm. In performing this multiplication, there are two main steps: First, we perform a one digit multiplication of each b_i with all of $a_{\ell-1}a_{\ell-2}\dots a_1a_0$, shifting one space to the left with each increasing i. In the second step, we add the rows we obtained using the addition from the previous subsection as a building block. For the first step, except in the case of $p = 2$ (where $b_i \in \{0, 1\}$, so the rows are $(a_{\ell-1} \cdot b_i)\dots(a_1 \cdot b_i)(a_0 \cdot b_i)$), this actually requires some computational effort because we have a carry r_i into the next digit. Similarly to Theorem 1, we can obtain the formula for this carry digit through a 3-fold Lagrange approximation over the variables a_i, b_i and r_i.

The second step consists of adding all the rows that we computed in the first step. We apply the improvement from [9] where we copy the digits of the upper row over the blank spaces on the right to the result, and apply a depth-optimal ordering in adding the rows. The exact formula can be found in the extended version [10], we instead illustrate our results here through Fig. 2.



(a) Additions (b) Multiplications (c) Depth

Fig. 2. Number of field additions, field multiplications and multiplicative depth for multiplying $x = 20/7000/10^7$ to a number of same size.

We can see that for additions and multiplications, the effort is lowest at $p = 2$ and increases with p, though there are again some sharp drops when the required number of digits decreases. As expectes, the depth issue has propagated from addition, which we used as a building block in multiplication: The best depth for $x = 20$ would be $p = 23$, the best depth for $x = 7000$ would be $p = 89$, and the best depth for $x = 10^7$ would be $p = 59$. We would like to point out that these values are not the same values that were optimal for addition (e.g., $p = 89$ is far from optimal for adding $x = 7000$) - thus, if one were to use depth as the sole metric, the optimal choice of p not only depends on the size of the numbers one is working with, but also on the number of additions vs. multiplications one wants to perform on these inputs. In the context of outsourced information, it is also important to note that optimizing the choice of p in this way could leak unwanted information, depending on the specific outsourcing scenario.

6 Using $GF(p^k)$ as Encoding Base

We now generalize our analysis to arbitrary finite fields as encoding bases. Much in the same way as in Sects. 4 and 5, we have also analyzed the effort incurred when using $GF(p^k)$ for a prime p and a $k > 1$ as an encoding base. First, recall that $GF(p^k) \cong \mathbb{Z}_p[X]/(f(x))$ with $f(x)$ irreducible of degree k. We embed a decimal number between 0 and $p^k - 1$ into $GF(p^k)$, whose elements are polynomials over $\mathbb{Z}_p$, through the insertion homomorphism: The element $a = \sum_{i=0}^{k-1} \alpha_i X^i \in GF(p^k)$ (with $\alpha_i \in \mathbb{Z}_p$) encodes the number $\tilde{a} = \sum_{i=0}^{k-1} \alpha_i p^i \in \mathbb{N}$. Generalizing this to numbers larger than $p^k - 1$ is straightforward: We will represent a number $\tilde{a} = \sum_{j=0}^{n} \tilde{a_j} (p^k)^j \in \mathbb{N}$ as $a_n a_{n-1} \ldots a_1 a_0$ where $a_j \in GF(p^k)$.

We now analyze the effort of adding two natural numbers in this encoding. Intuitively, we do not expect this to perform better than the encoding through $\mathbb{Z}_p$: The carry bit should roughly have the same effort as for $\mathbb{Z}_{p'}$ with p' of size comparable to p^k, but the addition is now more complicated. Concretely, the native addition of $GF(p^k)$ is that of $(\mathbb{Z}_p)^k$, i.e., it is done component-wise with no carry-over into other components, whereas we would need the addition of $\mathbb{Z}_{p^k}$ to natively support our encoding. Thus, we must emulate the addition $c_i = a_i + b_i + r_i$ in the same way as we compute the carry bit, so we expect a similar effort here and at least double the effort compared to $\mathbb{Z}_{p'}$ in total.

The results of this analysis are presented below – the detailed computation can be found in the extended version of this paper [10]. **To add two natural numbers of lenght $\ell \geq 2$, we have the following effort:**

- **Field additions:** $= (3\ell - 1) \cdot p^{2k} + (6\ell - 4) \cdot p^k - 6\ell$
- **Multiplications:** $= (6\ell - 2) \cdot p^k \cdot \log_2(p^k) + (4\ell - 2) \cdot p^k - 2\ell \cdot \log_2(p^k) - 11\ell + 3$
- **Constant multiplications:** $= (2\ell - 1) \cdot p^{2k} + \ell - 1$
- **Multiplicative depth:** $\lceil \log_2(p^k - 1) \rceil + \ell$

The case where the inputs are have only one digit is again slightly less expensive and has been omitted for brevity. We now compare the calculated effort to:

1. Encoding the number in base p instead of p^k and performing the addition.
2. Encoding the number in base p' with p' close to p^k.

Figure 3 shows the effort of adding two numbers of same size ($x = 20/7000/10^7$) in p^k-adic encoding for p^k up to 1000. Blue crosses are $\mathbb{Z}_p$, pink circles p^2, yellow triangles p^3, and the black square groups all bases p^k with $k \geq 4$, since the primes p with $p^k \leq 1000$ for increasing k become very few. We see that the p^k-encoding performs poorly regarding all metrics, and using $\mathbb{Z}_p$ as an encoding base is the better choice. Recall from Sect. 5.1 that the smaller the encoding base p for a plaintext space of $\mathbb{Z}_p$, the smaller the cost in terms of ciphertext additions and multiplications, and that the optimal base in terms of multiplicative depth varies. However, the factor that induces this variation is the required encoding length, and since we can choose a prime p' close to p^k (thus requiring roughly the same length) which requires much less effort as shown in Fig. 3, there is no case where choosing p^k as an encoding base with $k > 1$ brings any benefit, so we do not continue with its analysis.

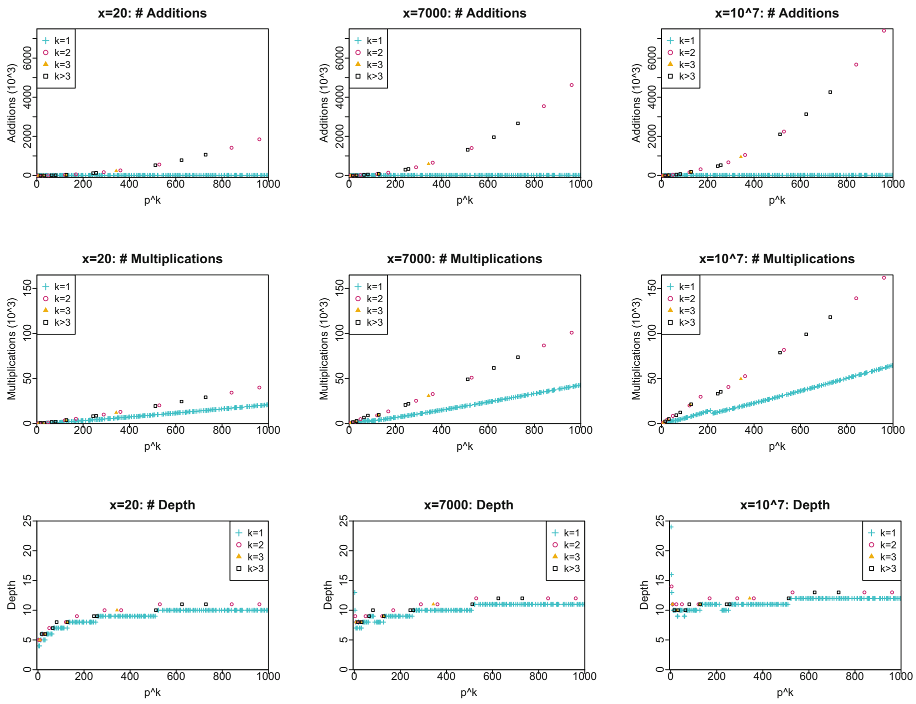


Fig. 3. Number of field additions, field multiplications and multiplicative depth for multiplying $x = 20$ (first column)/$x = 7000$ (second column)/$x = 10^7$ (right column) to a number of same size for encoding base $GF(p^k)$.

7 Rational Numbers and Integers

7.1 Representing Rational Numbers by Scaling

Let the encoding base p be an arbitrary prime. Given a rational number that we wish to encode (and assuming for the moment that negative numbers are no problem), we need to transform this rational into an integer, which we can then encode p-adically in the next step. The most straightforward approach is to introduce a scaling factor by picking a precision (i.e., there are σ p-adic digits after the point) σ with which we want to work with in the following and multiplying the rational with p^σ, rounding to obtain an integer to encode.

The importance of choosing the scaling factor as a power of p rather than any other number is as follows: Suppose that we have two rational numbers A and B which we scale and round to $\tilde{A} = A \cdot p^\sigma$ and $\tilde{B} = B \cdot p^\sigma$. After encoding and encrypting the individual digits, multiplying yields $\tilde{A} \cdot \tilde{B} = A \cdot B \cdot p^{2\sigma}$. Thus, after decrypting, the data owner needs to know what number to divide the result by, leaking unwanted information about the function that was applied, which might be the computing party's secret. Also the required number of digits increases to accommodate the extra precision digits, making all future computations less efficient. However, if we have a number in p-adic encoding, deleting the last σ digits corresponds to dividing by p^σ and truncating the result. This way, using p^σ as the scaling factor, the computing party can delete the σ least significant digits after each multiplication and thus keep the precision at a constant σ bits, increasing efficiency (by using less digits) and privacy (because the data owner now divides the result by p^σ regardless of the function that was applied).

7.2 Encoding Integers

Having seen how to transform rationals into integers, we need to incorporate negative numbers into our p-adic encoding. Generalizing from $p = 2$, for which these encodings are well known, we investigate two main approaches: p's Complement and Sign-Magnitude. Note that this question has been extensively studied in [9] but for the case $p = 2$ only. We state shortly how the results extend to the case $p > 2$, with a more detailed analysis in the extended version [10].

p's Complement: In this encoding, elements have the form $a_n \dots a_0$ with $a_i \in \{0, 1, \dots, p-1\}$ for $i = 0, \dots, n-1$, and $a_n \in \{0, 1\}$, where $x = -a_n \cdot p^n + \sum_{i=0}^{n-1} a_i \cdot p^i$. This means that the first digit encodes either 0 or $-p^n$ and the following digits correspond to the "normal" p-adic encoding.

Addition: The effort of adding two numbers in p's Complement encoding is comparable to twice the effort for adding two natural numbers derived in Sect. 5.1, except that the depth is twice as large. For $p = 2$, it is almost exactly the same effort as adding two natural numbers in binary encoding.

Multiplication: Multiplication p's Complement roughly requires the same effort as a natural number with twice as many digits, i.e., multiplying a natural number

x with ℓ digits in p's Complement encoding requires roughly the same effort as multiplying x^2 (which has 2ℓ digits) in "regular" p-adic encoding.

Sign-Magnitude: For the second encoding that we consider, Sign-Magnitude, the absolute value of the number is encoded p-adically as a natural number, and there is an extra digit (the MSB) which determines the sign. Concretely, elements in this encoding have the form $a_n a_{n-1} \dots a_1 a_0$ with $a_i \in \{0, 1, \dots, p-1\}$ for $i = 0, \dots, n-1$, and $a_n \in \{0, 1\}$, where $x = (-1)^{a_n} \cdot \sum_{i=0}^{n-1} a_i \cdot p^i$. It is easy to see that for positive numbers, this encoding is the same as p's complement. This encoding suffers from having two representations of 0: $00 \dots 0$ and $100 \dots 0$.

Addition: Adding two numbers in Sign-Magnitude encoding costs roughly one p-adic addition, 2 comparisons (each costing about three additions) and 4 subtractions (about the same cost as addition). This yields 11 additions in "regular" p-adic encoding and is significantly more costly than $p's$ Complement encoding.

Multiplication: The effort of multiplying two numbers in Sign-Magnitude encoding is roughly the same as multiplying them with "regular" p-adic encoding.

Hybrid Encoding: We see that the choice of encoding can make a big difference in performance. As p's Complement addition is more efficient than Sign-Magnitude, but the latter is more efficient for multiplication, a hybrid approach like in [9] would be the best choice: One does all additions in p's Complement, and for multiplication switches the encoding to Sign-Magnitude. Using this, one can have roughly the same operation cost as for natural numbers in p-adic encoding (slightly more for additions), plus the cost of switching between encodings, which is roughly that of one p-adic addition. Of course, since this already holds true for natural numbers in p-adic encoding, the choice $p = 2$ by far incurs the least amount of field additions and multiplications in these two encodings and the hybrid encoding also, while the optimal depth choice remains variable.

References

1. Arita, S., Nakasato, S.: Fully homomorphic encryption for point numbers. IACR Cryptology ePrint Archive 2016/402 (2016)
2. Bonte, C., Bootland, C., Bos, J.W., Castryck, W., Iliashenko, I., Vercauteren, F.: Faster homomorphic function evaluation using non-integral base encoding. IACR Cryptology ePrint Archive 2017/333 (2017)
3. Chen, Y., Gong, G.: Integer arithmetic over ciphertext and homomorphic data aggregation. In: CNS (2015)
4. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. IACR Cryptology ePrint Archive 2016/421 (2016)
5. Chung, H., Kim, M.: Encoding rational numbers for FHE-based applications. IACR Cryptology ePrint Archive 2016/344(2016)
6. Costache, A., Smart, N.P., Vivek, S., Waller, A.: Fixed point arithmetic in SHE scheme. IACR Cryptology ePrint Archive 2016/250 (2016)
7. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 24–43. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_2

8. Dowlin, N., Gilad-Bachrach, R., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: Manual for using homomorphic encryption for bioinformatics. Technical report. MSR-TR-2015-87, Microsoft Research (2015)
9. Jäschke, A., Armknecht, F.: Accelerating homomorphic computations on rational numbers. In: Manulis, M., Sadeghi, A.-R., Schneider, S. (eds.) ACNS 2016. LNCS, vol. 9696, pp. 405–423. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39555-5_22
10. Jäschke, A., Armknecht, F.: (Finite) field work: choosing the best encoding of numbers for FHE Computation. IACR Cryptology ePrint Archive 2017/582 (2017)
11. Kim, E., Tibouchi, M.: FHE over the integers and modular arithmetic circuits. In: CANS, pp. 435–450 (2016)
12. Nuida, K., Kurosawa, K.: (Batch) fully homomorphic encryption over integers for non-binary message spaces. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 537–555. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_21
13. Xu, C., Chen, J., Wu, W., Feng, Y.: Homomorphically encrypted arithmetic operations over the integer ring. In: Bao, F., Chen, L., Deng, R.H., Wang, G. (eds.) ISPEC 2016. LNCS, vol. 10060, pp. 167–181. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49151-6_12