PIE: \( p \)-adic Encoding for High-Precision Arithmetic using Homomorphic Encryption

Gaetan Delavignette \(^1\) Luke Harmon \(^1\) Arnab Roy \(^1\) David Silva \(^1\)

\(^1\)Algemetic

Motivation

Rational numbers are frequently used in many real-world settings such as finance, health care, and business intelligence. Further, these data must often be composed in ways which require addition and multiplication of rational numbers.

**p-adic numbers, plaintexts, and the encoding function**

**p-adic representation:** If \( q \in \mathbb{Q} \) and \( p \) is a prime then we have

\[
\frac{a}{b} = \sum_{j} a_j p^j + a_{-1} p^{-1} + \ldots
\]

where \( 0 \leq a_j < p \) and \( a_j \in \mathbb{Z} \). An \( \alpha \)-segment \( p \)-adic representation, a.k.a. Hensel code, simply truncates the above sum after \( j = \alpha - 1 \).

A \( p \)-adic integer is a rational which can be written in the form

\[
\frac{a}{b} = \sum_{j} a_j p^j, \quad \text{for } 0 \geq a_j \text{ and } g(y, x) = g(y, p) = 1
\]

Given a prime \( p \) and an integer \( r \geq 1 \), let \( N = \frac{1}{p^r} \). The Farey rationals are defined as

\[
\mathcal{F}_N = \left\{ \frac{a}{b} \mid 0 < a \leq b \leq N, 1 \leq b \leq N, g(y, x) = g(y, p) = 1 \right\}
\]

\( \mathcal{F}_N \) is a set of \( p \)-adic integers. Given \( a, x, y \in \mathbb{Z} \),

\[
\text{MEEA}(a, x, y) = \left( (-1)^i x_j, (-1)^i y_j \right), i \geq 0
\]

where \( x, y \) are generated by the extended Euclidean algorithm (EEA). The MEEA simply stops EEA early (once \( |y| \leq N \)).

An homomorphic mapping: The injective mapping \( H_p : \mathcal{F}_N \rightarrow \mathbb{Z}/p\mathbb{Z} \) and its inverse are defined as

\[
H_p \left( \left( \frac{a}{b} \right) \right) = a \mod p
\]

\[
H_p^{-1}(hk) = \text{MEEA}(k, x, y)
\]

**PIE Encoder**

Let \( y \) be a positive integer and \( N = \sqrt{v(y - 1)/2} \). \( \mathcal{F}_N \) is the input space. Since it is not closed under \( + \) and \( \cdot \), we define \( \mathcal{D}_N = \{ x/y \mid |x|, |y| \leq M \} \subset \mathcal{F}_N \) as the message space.

Encoding and decoding are defined as follows:

**PIE Encode:**

\[
\text{PIE Encode} \left( \left( \frac{a}{b} \right) \right) \text{ For } \frac{a}{b} \in \mathcal{F}_N, \text{ output } H_p \left( \frac{a}{b} \right) \in \mathbb{Z}/q\mathbb{Z}.
\]

**PIE Decode:**

\[
\text{PIE Decode} \left( \frac{a}{b} \right) \text{ For } \frac{a}{b} \in \mathbb{Z}/q\mathbb{Z}, \text{ output } H_p^{-1}(a/b) \in \mathcal{F}_N,
\]

The smaller \( M \) is relative to \( N \), the deeper the circuits with which PIE is compatible.

**PIE with an AGCD-based Batch FHE**

We attach PIE to the batch integer FHE scheme (IDGHV) from [2].

Choose the public parameters \( Q_1, \ldots, Q_t \) of IDGHV to be distinct odd primes, let \( y = \prod_t Q_t \), and \( N = \sqrt{v(y - 1)/2} \). We encode and decode as follows:

**IDGHV Encode:**

For \( m \in \mathbb{Z}/y\mathbb{Z} \), let \( \text{PIE Encode}(m) \) mod \( Q_1, \ldots, Q_t \) and \( \text{PIE Encode}(m) \) mod \( Q_t \).

**IDGHV Decode:**

For \( h_1, \ldots, h_t \in \mathbb{Z}/Q_i\mathbb{Z} \times \cdots \times \mathbb{Z}/Q_t\mathbb{Z} \), compute \( h = \text{CRT}(h_1, \ldots, h_t) \), then output PIE Decode(\( h \)).

The set \( \mathbb{P}_\mathcal{F} \) is the set of multivariate polynomials with integer coefficients, degree \( d \), and \( d_0 \) norms at most \( t \). Encoding correctness, i.e. \( \text{Decide}(\text{Encode}(m)) = m \), is ensured by choosing \( M \leq (N/y)^{1/2} \).

**Library and Demo**

PIE is implemented in C++, using NTL and GMP for large integer, vector, and polynomial arithmetic, and Bazel as the build system, and is available at https://github.com/Algemetic/pie-cpp. PIE can be built as a stand-alone fractional encoding library or in conjunction with either IDGHV or ModFV. The implementation is currently not optimized for performance, therefore the numbers below aim to highlight the overhead of adding PIE to an HE scheme.

**References**


**Table 1:** Comparison of input space sizes for PIE and ModFV encoder when \( p^r + 1 \) is prime.

<table>
<thead>
<tr>
<th>( p )</th>
<th>PIE</th>
<th>CLFx</th>
<th>PIE/ModFV</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>12</td>
<td>442765</td>
<td>372573</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>328324</td>
<td>97056</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>479890</td>
<td>908900</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>6 × 10^4</td>
<td>2.2 × 10^4</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>1.2 × 10^5</td>
<td>1.9 × 10^5</td>
</tr>
<tr>
<td>13</td>
<td>10</td>
<td>4.4 × 10^5</td>
<td>1.1 × 10^6</td>
</tr>
</tbody>
</table>

**Table 2:** Comparison of input space sizes for PIE and ModFV encoder when \( p^r + 1 \) is composite.

<table>
<thead>
<tr>
<th>( p )</th>
<th>PIE</th>
<th>ModFV</th>
<th>PIE/ModFV</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>15624</td>
<td>1282</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>1544</td>
<td>1182</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>1544</td>
<td>1182</td>
<td>13</td>
</tr>
</tbody>
</table>

**Table 3:** Example of encoding times (for IDGHV scheme).

<table>
<thead>
<tr>
<th>( \lambda )</th>
<th>( \ell )</th>
<th>( \mu )</th>
<th>( \tau )</th>
<th>Encoding time</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>60</td>
<td>3282</td>
<td>5.3</td>
<td>0.006ms</td>
</tr>
<tr>
<td>52</td>
<td>32</td>
<td>1158</td>
<td>9.1</td>
<td>0.0078ms</td>
</tr>
</tbody>
</table>

**Figure 1:** Attaching PIE to an FHE scheme.

**Figure 2:** Encoding times for IDGHV (in ms).