Building Algorithm-Hiding FHE Systems from Exotic Number Representations

P. Martins\textsuperscript{1} L. Sousa\textsuperscript{1}

\textsuperscript{1}INESC-ID
Instituto Superior Técnico, Univ. Lisboa

Workshop on Randomness and Arithmetics for Cryptography on Hardware
Table of Contents

Motivation

Background

Proposed Solution

Experimental Results

Related Art

Conclusion
Motivation
Motivation

- Data disclosure is prevented
- What about algorithm disclosure?
Table of Contents

Motivation

Background

Proposed Solution

Experimental Results

Related Art

Conclusion
Solution #1

Describe GP-CPU as Homomorphic Circuit

Convert Algorithm to Instruction Memory

Homomorphically Evaluate GP-CPU

Solution #1

- The evaluator does not know which instruction is being executed
- All the CPU circuitry needs to be evaluated at each cycle
- Including memory accesses, ALU operations, etc
Solution #1

- The evaluator does not know which instruction is being executed
- All the CPU circuitry needs to be evaluated at each cycle
- Including memory accesses, ALU operations, etc

⇒ Impractical
**BGV**

- **Ring:** \( R = \mathbb{Z}[X]/(\phi_m(X)) \)
  \( \phi_m(X) \) is a cyclotomic polynomial of degree \( \varphi(m) \)

- **Ciphertexts:** \( c_0 + c_1 Y \in R_q[Y] \)

- **Decryption:** \([c_0 + c_1 s]_q = [[m]_2 + 2v]_q\)
  \( m \in R_2 \)

- **Addition:** \((c_0 + c'_0) + (c_1 + c'_1) Y\)
  evaluated at \( Y = s \) leads to \( \approx [[m + m']_2 + 2(v + v')]_q \)

Multiplication: \((c_0 + c_1 Y) \times (c'_0 + c'_1 Y) = \) \(ct_{mult,0} + c_{mult,1} Y + ct_{mult,2} Y^2\)
evaluated at \(Y = s\) leads to \(\approx [\left[ m \times m' \right]_2 + 2v'']_q\)

Relinearisation: Multiply \(ct_{mult,2}\) by pseudo-encryption of \(s^2\) and add to \((ct_{mult,0}, ct_{mult,1})\)

Modulus-switching:

\[
\delta_i \leftarrow 2 \cdot \left[ -ct_{mult,i}/2 \right]_{q/q'} \text{ for } i = 0, 1
\]
\[
ct \leftarrow \left( \left[ q'/q \cdot (ct_{mult,0} + \delta_0) \right]_{q'}, \left[ q'/q \cdot (ct_{mult,1} + \delta_1) \right]_{q'} \right)
\]
Table of Contents

Motivation

Background

Proposed Solution

Experimental Results

Related Art

Conclusion
“Natural” Homomorphic Structure #1

- Binary plaintext space

\[ P = \mathbb{Z}[X]/(\phi_m(X), 2) \]

with \( \phi_m = F_0 \times \ldots \times F_{l-1} \mod 2 \)

- Exploit factorisation to encrypt multiple bits in a single ciphertext

- Bits \( m_0, \ldots, m_{l-1} \) are encoded as

\[ m_i = m(x) \mod (F_i(x), 2) \quad \forall 0 \leq i < l \]

- Hom. additions and multiplications operate on them in parallel
“Natural” Homomorphic Structure #1

- Represent $x \in [0, 1]$ as $x_0, \ldots, x_{l-1} \in \{0, 1\}$ s.t.
  \[ P(x_i = 1) = x \]

- Batch-encrypt $x_0, \ldots, x_{l-1}$

- Coefficient-wise multiplications and scaled additions
  \[ z_i = x_i \land y_i \Rightarrow z = xy \]
  \[ z_i = ((1 \oplus s_i) \land x_i) \oplus (s_i \land y_i) \Rightarrow z = (1 - s)x + sy \]

“Natural” Homomorphic Structure #1

Require: \( B(x) = \sum_{i=0}^{d} \binom{d}{i} b_i x^i (1 - x)^{d-i} \)

Require: \( x_0 \)

1: for \( i \in \{0, \ldots, d\} \) do
2: \( b_i^{(0)} := b_i \)
3: end for
4: for \( j \in \{1, \ldots, d\} \) do
5: for \( i \in \{0, \ldots, d - j\} \) do
6: \( b_i^{(j)} := b_i^{(j-1)} (1 - x_0) + b_{i+1}^{(j-1)} x_0 \)
7: end for
8: end for
9: return \( B(x_0) = b_0^{(d)} \)

De Casteljau’s algorithm for the evaluation of a polynomial in Bernstein form
Modify BGV with the following decryption

\[ [c_0 + c_1 s]_q = [m + v]_q \]

A number \( x \in \mathbb{R} \) is represented as a polynomial

\[ x = \lfloor \Delta x \rfloor + v \]

After multiplications, rescale

\[ ct \leftarrow \left( \left\lfloor \left\lfloor q'/q \cdot ct_{mult,0} \right\rfloor q' \right\rfloor, \left\lfloor \left\lfloor q'/q \cdot ct_{mult,1} \right\rfloor q' \right\rfloor \right) \]

“Natural” Homomorphic Structure #2

Require: \( P(x) = \sum_{i=0}^{d} a_i x^i \)

Require: \( x_0 \)

1: \( s := a_d \)
2: \( \text{for } i \in \{d - 1, \ldots, 0\} \text{ do} \)
3: \( s := a_i + x_0 s \)
4: \( \text{end for} \)
5: \( \text{return } P(x_0) = s \)

Horner’s method for the evaluation of a polynomial in power form
Approximate continuous functions with Bernstein polynomials through Weierstrass theorem

If necessary, convert Bernstein polynomials to power form

Factorise multivariate polynomials into univariate polynomials

Use de Casteljau algorithm or Horner’s method
Approximate continuous functions with Bernstein polynomials through Weierstrass theorem

$$\beta_{f,k_1,...,k_m}^{(n_1,...,n_m)} = f \left( \frac{k_1}{n_1}, \ldots, \frac{k_m}{n_m} \right)$$

$$B_{f}^{(n_1,...,n_m)}(x_1, \ldots, x_m) = \sum_{0 \leq k_l \leq n_l \atop l \in \{1,\ldots,m\}} \beta_{f,k_1,...,k_m}^{(n_1,...,n_m)} \prod_{j=1}^{m} \binom{n_j}{k_j} x_j^{k_j} (1-x_j)^{n_j-k_j}$$
If necessary, convert Bernstein polynomials to power form

\[ x_{j_1} \ldots x_{j_m} = \sum_{k_1=j_1}^{n_1} \binom{k_1}{j_1} \binom{n_1}{k_1} x_1^{k_1} (1 - x_1)^{n_1-k_1} \times \]

\[ \ldots \times \sum_{k_m=j_m}^{n_m} \binom{k_m}{j_m} \binom{n_m}{k_m} x_m^{k_m} (1 - x_m)^{n_m-k_m} = \]

\[ \sum_{j_l \leq k_l \leq n_l} \prod_{h=1}^{m} \binom{k_h}{j_h} \binom{n_h}{k_h} x_h^{k_h} (1 - x_h)^{n_h-k_h} \]
Factorise multivariate polynomials into univariate polynomials

\[ B^{(n_1,\ldots,n_m)}_f(x_1,\ldots,x_m) = \sum_{k_1=0}^{n_1} \binom{n_1}{k_1} x_1^{k_1} (1 - x_1)^{n_1-k_1} \left( \sum_{k_2=0}^{n_2} \binom{n_2}{k_2} x_2^{k_2} (1 - x_2)^{n_2-k_2} \right) \ldots \]

\[ P(x_1,\ldots,x_m) = \sum_{k_1=0}^{n_1} x_1^{k_1} \left( \sum_{k_2=0}^{n_2} x_2^{k_2} \ldots \left( \sum_{k_m=0}^{n_m} \binom{n_m}{k_m} x_m^{k_m} (1 - x_m)^{n_m-k_m} \right) \right) \ldots \]
Proposed Computing Model

\[
\begin{align*}
\text{Encrypt}_E \left( \beta_{f,k_1,...,k_m}^{(n_1,...,n_m)} \right) \\
\text{or} \\
\text{Encrypt}_E \left( \alpha_{f,k_1,...,k_m}^{(n_1,...,n_m)} \right)
\end{align*}
\]

Homomorphic Evaluator
de Casteljau or Horner

\[
\text{Encrypt}_E \left( f \left( x_1, \ldots, x_m \right) \right)
\]
Table of Contents

Motivation

Background

Proposed Solution

Experimental Results

Related Art

Conclusion
Example #1

**Require:** \( z \in \mathbb{R}^K \)

1. Sort \((z_1, \ldots, z_K)\) as \((z^{(1)}, \ldots, z^{(K)})\) s.t. \(z^{(1)} \geq \ldots \geq z^{(K)}\)

2. \(k(z) := \max \left\{ k \in \{1, \ldots, K\} \mid 1 + kz^{(k)} > \sum_{j \leq k} z^{(j)} \right\} \)

3. \(\tau(z) := \frac{(\sum_{j \leq k(z)} z^{(j)}) - 1}{k(z)}\)

4. **return** \( p \) s.t. \(p_i := \max(0, z_i - \tau(z))\)

Sparsemax function for mapping scores to probabilities
### Example #1

<table>
<thead>
<tr>
<th>Function</th>
<th>Scheme</th>
<th># slots</th>
<th>(n_1)</th>
<th>(n_2)</th>
<th>(m)</th>
<th>(\log_2 q)</th>
<th>MAE</th>
<th>Sequential Execution Time [s]</th>
<th>Parallel Execution Time [s]</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{sparsemax}_1(x_1, 0))</td>
<td>Fixed-point</td>
<td>5</td>
<td>1</td>
<td>2(^{15})</td>
<td>744</td>
<td>0.0843</td>
<td>0.489</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(\text{sparsemax}_1(x_1, 0))</td>
<td>Fixed-point</td>
<td>10</td>
<td>1</td>
<td>2(^{15})</td>
<td>744</td>
<td>0.0495</td>
<td>0.689</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(\text{sparsemax}_1(x_1, 0))</td>
<td>Fixed-point</td>
<td>15</td>
<td>1</td>
<td>2(^{16})</td>
<td>1550</td>
<td>0.0336</td>
<td>9.00</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(\text{sparsemax}_1(x_1, x_2, 0))</td>
<td>Fixed-point</td>
<td>2</td>
<td>2</td>
<td>2(^{15})</td>
<td>744</td>
<td>0.181</td>
<td>0.902</td>
<td>0.543</td>
<td>1.7</td>
<td></td>
</tr>
<tr>
<td>(\text{sparsemax}_1(x_1, x_2, 0))</td>
<td>Fixed-point</td>
<td>3</td>
<td>3</td>
<td>2(^{15})</td>
<td>744</td>
<td>0.133</td>
<td>1.57</td>
<td>0.687</td>
<td>2.3</td>
<td></td>
</tr>
<tr>
<td>(\text{sparsemax}_1(x_1, x_2, 0))</td>
<td>Fixed-point</td>
<td>4</td>
<td>4</td>
<td>2(^{16})</td>
<td>1550</td>
<td>0.120</td>
<td>20.7</td>
<td>6.87</td>
<td>3.0</td>
<td></td>
</tr>
<tr>
<td>(\text{sparsemax}_1(x_1, 0))</td>
<td>Stochastic</td>
<td>630</td>
<td>5</td>
<td>8191</td>
<td>327</td>
<td>0.104</td>
<td>0.409</td>
<td>0.272</td>
<td>1.5</td>
<td></td>
</tr>
<tr>
<td>(\text{sparsemax}_1(x_1, 0))</td>
<td>Stochastic</td>
<td>1024</td>
<td>10</td>
<td>21845</td>
<td>1440</td>
<td>0.063</td>
<td>16.2</td>
<td>6.40</td>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>(\text{sparsemax}_1(x_1, 0))</td>
<td>Stochastic</td>
<td>2160</td>
<td>15</td>
<td>55831</td>
<td>2592</td>
<td>0.036</td>
<td>83.0</td>
<td>19.5</td>
<td>4.3</td>
<td></td>
</tr>
<tr>
<td>(\text{sparsemax}_1(x_1, x_2, 0))</td>
<td>Stochastic</td>
<td>630</td>
<td>2</td>
<td>8191</td>
<td>327</td>
<td>0.151</td>
<td>0.301</td>
<td>0.254</td>
<td>1.1</td>
<td></td>
</tr>
<tr>
<td>(\text{sparsemax}_1(x_1, x_2, 0))</td>
<td>Stochastic</td>
<td>1024</td>
<td>3</td>
<td>21845</td>
<td>985</td>
<td>0.129</td>
<td>9.46</td>
<td>3.58</td>
<td>2.6</td>
<td></td>
</tr>
<tr>
<td>(\text{sparsemax}_1(x_1, x_2, 0))</td>
<td>Stochastic</td>
<td>2160</td>
<td>4</td>
<td>55831</td>
<td>2592</td>
<td>0.112</td>
<td>39.6</td>
<td>9.78</td>
<td>4.0</td>
<td></td>
</tr>
</tbody>
</table>

The functions \(\text{sparsemax}_1(x_1, 0)\) and \(\text{sparsemax}_1(x_1, x_2, 0)\) were approximated and homomorphically evaluated on an i7-5960X, using both a fixed-point approach with Horner’s scheme and a stochastic number representation with de Casteljau’s algorithm.
Example #2

Blending:
(a) Clear  (b) Fixed-point  (c) Stochastic

Grey Stretching:
(a) Clear  (b) Fixed-point  (c) Stochastic
### Example #2

<table>
<thead>
<tr>
<th>System</th>
<th>Encryption [s] Intel / Arm</th>
<th>Filter [s] Intel</th>
<th>Decryption [s] Intel / Arm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grey Stretching – Fixed-point</td>
<td>52.5 / 685</td>
<td>341</td>
<td>6.9 / 134</td>
</tr>
<tr>
<td>Blending – Fixed-point</td>
<td>52.7 / 684</td>
<td>885</td>
<td>5.3 / 88</td>
</tr>
<tr>
<td>Grey Stretching – Stochastic</td>
<td>34.5 / 914</td>
<td>1340</td>
<td>61.7 / 1172</td>
</tr>
<tr>
<td>Blending – Stochastic</td>
<td>47.7 / 1273</td>
<td>2103</td>
<td>89.4 / 1468</td>
</tr>
<tr>
<td>Grey Stretching – Floating-point</td>
<td>324 / 7935</td>
<td>95.9</td>
<td>92.7 / 2630</td>
</tr>
</tbody>
</table>

Average execution time for homomorphic image processing operations on an i7-5960X (Intel) and on a Cortex-A53 (Arm). The last implementation corresponds to an adaption of † to the proposed system. † uses the Paillier cryptosystem

Related Art

<table>
<thead>
<tr>
<th>Computing Model</th>
<th>Performance</th>
<th>Development Effort</th>
<th>Scope</th>
<th>Privacy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional</td>
<td>Directly exploits CPU architecture</td>
<td>Traditional programming techniques</td>
<td>Supports any application</td>
<td>Vulnerable to attacks like Meltdown and Spectre</td>
</tr>
<tr>
<td>PHE libraries</td>
<td>Overhead associated with PHE</td>
<td>Intricate development. Requires strong familiarity with PHE</td>
<td>Limited support of applications</td>
<td>Hides data</td>
</tr>
<tr>
<td>FHE w/ application specific circuits</td>
<td>Overhead associated with FHE</td>
<td>Intricate development. Requires strong familiarity with FHE</td>
<td>Supports most applications</td>
<td>Hides data</td>
</tr>
<tr>
<td>Proposed model</td>
<td>Limited set of FHE operations</td>
<td>Traditional programming techniques</td>
<td>Continuous functions</td>
<td>Hides data and algorithm</td>
</tr>
<tr>
<td>FHE w/ encrypted computer architecture</td>
<td>Impractical</td>
<td>Halting problem may cause development issues</td>
<td>Supports most applications</td>
<td>Hides data and algorithm</td>
</tr>
</tbody>
</table>

Best: \(\text{FHE w/ encrypted computer architecture}\)

Worst: \(\text{FHE w/ application specific circuits}\)
Conclusion

▶ Current cloud computing models vulnerable to data and algorithm disclosure

▶ While FHE prevents data leaking, achieving algorithm secrecy has been impractical so far

▶ Herein, we focus on a wide range of functions whose approximations can be efficiently evaluated with homomorphic operations

▶ All approximations are evaluated in the same manner ⇒ an evaluator has no way to distinguish them
Thank you!

Any questions?