Fully Homomorphic Encryption

Amit Datta
08CS1045

Mentored by
Prof. Debdeep Mukhopadhyay
Outline

- Motivation
- A naïve scheme and its problems
- Existing Scheme and its implementation
- Problems
- Future Work
The Goal of FHE

- I want to delegate processing of my data, without giving away access to it.
  - Craig Gentry (2009)
Application 1 – Cloud Computing

- Data stored on cloud in encrypted form
- You want to perform SECRET operations on the data
- Encrypt simple queries to _queries_
- Send _queries_ to cloud
- Cloud performs _queries_ on encrypted data and sends back encrypted results
- Decrypt them to get actual results
Application 2 – Private Google Search

- You don’t want Google to know your SECRET queries

![Google Search Form](image)

<table>
<thead>
<tr>
<th>How can one destroy the Google headquarters?</th>
</tr>
</thead>
</table>

Google.co.in offered in: Hindi Bengali Telugu Marathi Tamil Gujarati Kannada Malayalam Punjabi
Application 2 – Private Google Search

- You don’t want Google to know your SECRET queries
- Submit encrypted queries
- Get encrypted results
- Decrypt results
Our Goal(s)

- Perform operations of data without knowing the contents **EFFICIENTLY**

- Performing attacks on the existing scheme, especially **SIDE-CHANNEL ATTACKS**.
A simple scheme

- Shared secret key: odd number p
- To encrypt a bit m in \{0,1\}:
  - Choose at random small r, large q
  - Output \( c = m + 2r + pq \)
    - \( m = \text{LSB of distance to nearest multiple of } p \)
- To decrypt c:
  - Output \( m = (c \mod p) \mod 2 \)
A simple scheme

- Shared secret key: odd number \( p = 101 \)
- To encrypt a bit \( m \) in \( \{0,1\} \):
- Choose at random small \( r \), large \( q \)
- Output \( c = m + 2r + pq \)

\[ m = \text{LSB of distance to nearest multiple of } p \]

- To decrypt \( c \):
- Output \( m = (c \mod p) \mod 2 \)
A simple scheme

- Shared secret key: odd number $p (=101)$
- To encrypt a bit $m$ in $\{0,1\}$: (say $m=1$)
- Choose at random small $r$, large $q$
- Output $c = m + 2r + pq$
  
  $m = \text{LSB of distance to nearest multiple of } p$

- To decrypt $c$:
- Output $m = (c \mod p) \mod 2$
A simple scheme

- Shared secret key: odd number $p (=101)$
- To encrypt a bit $m$ in $\{0,1\}$: (say $m=1$)
- Choose at random small $r (=5)$, large $q (=9)$
- Output $c = m^{\text{noise}} + 2r + pq$
  
  \[ m = \text{LSB of distance to nearest multiple of } p \]

- To decrypt $c$:
- Output $m = (c \mod p) \mod 2$
A simple scheme

- Shared secret key: odd number \( p = 101 \)
- To encrypt a bit \( m \) in \( \{0, 1\} \): (say \( m = 1 \))
- Choose at random small \( r = 5 \), large \( q = 9 \)
- Output \( c = m + 2r + pq = 1 + 10 + 909 = 920 \)
  \( m = \text{LSB of distance to nearest multiple of } p \)

- To decrypt \( c \):
- Output \( m = (c \mod p) \mod 2 \)
A simple scheme

- Shared secret key: odd number \( p = 101 \)
- To encrypt a bit \( m \) in \{0,1\}: (say \( m = 1 \))
  - Choose at random small \( r = 5 \), large \( q = 9 \)
  - Output \( c = m + 2r + pq = 1 + 10 + 909 = 920 \)
    - \( m \) = LSB of distance to nearest multiple of \( p \)
- To decrypt \( c \):
  - Output \( m = (c \mod p) \mod 2 = (920 \mod 101) \mod 2 = 11 \mod 2 = 1 \)
Homomorphism?

- \( c_1 = m_1 + 2r_1 + pq_1 \)  \( \quad \) \( c_2 = m_2 + 2r_2 + pq_2 \)

Noise

- \( c_1 + c_2 = (m_1 + m_2) + 2(r_1 + r_2) + p(q_1 + q_2) \)
- \( (c_1 + c_2 \mod p) \mod 2 = m_1 + m_2 \mod 2 = m_1 \text{ XOR } m_2 \)

Noise

- \( c_1 \cdot c_2 = (m_1 + 2r_1) \cdot (m_2 + 2r_2) + p(q') \)
- \( (c_1 \cdot c_2 \mod p) \mod 2 = m_1 \cdot m_2 \mod 2 = m_1 \text{ AND } m_2 \)
Homomorphism?

- $c_1 = m_1 + 2r_1 + pq_1$
- $c_2 = m_2 + 2r_2 + pq_2$

- $c_1 = 1 + 2.5 + 9.101$
- $c_2 = 1 + 2.7 + 8.101$

- $c_1 = 11 + 909$
- $c_2 = 15 + 808$

- $c_1 \cdot c_2 = 11.15 + 295.101 = 165 + 295.101$

- $c_1 \cdot c_2 \mod 101 = 165 \mod 101 = 64$
- $(c_1 \cdot c_2 \mod 101) \mod 2 = 0$ (Incorrect!)
A simple scheme

- Shared secret key: odd number $p (=101)$
- To encrypt a bit $m$ in $\{0,1\}$: (say $m=1$)
- Choose at random small $r (=5)$, large $q (=9)$
- Output $c = m + 2r + pq = 1 + 10 + 909 = 920$
  
  $m = \text{LSB of distance to nearest multiple of } p$

- To decrypt $c$:
- Output $m = (c \mod p) \mod 2$
  
  $= (920 \mod 101) \mod 2 = 11 \mod 2 = 1$
Noise Problem

- Problem arises when noise becomes comparable to p
- When this happens, cipher-texts could be decrypted, and again encrypted with fresh noise, which is always small
Noise Problem

- Problem arises when noise becomes comparable to $p$
- When this happens, cipher-texts could be decrypted, and again encrypted with fresh noise, which is always small

- Wouldn’t that compromise privacy?
Need: A Bootstrappable Scheme

- A scheme which can handle its own decryption function
- If such a scheme can be designed, cipher texts encrypted under one key, can be encrypted for another level with another key, and then one level of encryption removed
Need: A Bootstrappable Scheme

- A scheme which can handle its own decryption function
- If such a scheme can be designed, cipher texts encrypted under one key, can be encrypted for another level with another key, and then one level of encryption removed

- We will come back to this!
Gentry’s FHE scheme

- KeyGen($\lambda$)
- Encrypt($pk$, $m$)
- Decrypt($sk$, $m$)
- Evaluate($pk$, $f$, $c_1$, ..., $c_t$)
- Recrypt($pk_2$, $D_\epsilon$, $sk_1$, $c_1$)
Parameter Declaration

- Read security parameter $\lambda$
- Set $N \leftarrow \lambda$, $P \leftarrow \lambda^2$, $Q \leftarrow \lambda^5$
- Randomly select two integer parameters $0 < \alpha < \beta$
Gentry’s FHE scheme

- $\text{KeyGen}(\lambda)$
  - $\text{Encrypt}(pk, m)$
  - $\text{Decrypt}(sk, m)$
  - $\text{Evaluate}(pk, f, c_1, \ldots, c_t)$
  - $\text{Recrypt}(pk_2, D_\varepsilon, sk_1, c_1)$
KeyGen(λ)

- Generates pk, sk
- p is a random P-bit odd integer
- Generate a set \( y = \{ y_1, \ldots, y_\beta \} \): \( y_i \in [0, 2) \)
- For a sparse subset \( S \) of size \( \alpha \),
  \( \sum y_S = (1/p) \mod 2 \)
- \( sk \leftarrow s \), where \( s = \{0,1\}^\beta \) is an encoding of \( S \)
- \( pk \leftarrow (p, y) \)
Implementation Technique

- Structure **publicKey** defined with one integer \( p \) and an array \( (y) \) of reals for pk.
- Each element is subset solution is set at \( (1/p+2 \text{rand()} \mod \alpha)/\alpha \)
- Every other element of \( y \) is set randomly
Gentry’s FHE scheme

- $\text{KeyGen}(\lambda)$
- $\text{Encrypt}(pk, m)$
- $\text{Decrypt}(sk, m)$
- $\text{Evaluate}(pk, f, c_1, \ldots, c_t)$
- $\text{Recrypt}(pk_2, D_\epsilon, sk_1, c_1)$
Encrypt(pk, m)

- Generate an N-bit integer m’ such that m’ = m mod 2
- Generate a random Q-bit integer q
- Set c = m’ + (pk.p)*q
- Generate a set \( z : z_i \leftarrow c*y_i \mod 2 \)
- Return \( c \leftarrow (c, z) \)
Implementation Technique

- Required a mod2 function, which can compute values of reals modulo 2.
- Necessary for post-processing \( y \) to compute \( z \).
Gentry’s FHE scheme

- KeyGen($\lambda$)
- Encrypt(pk, m)
- Decrypt(sk, m)

- Evaluate(pk, f, c_1, ..., c_t)
- Recrypt(pk_2, D_\epsilon, sk_1, c_1)
Decrypt(sk, c)

- To return \((c \mod p) \mod 2\)
- Equivalent to \(\text{LSB}(c) \oplus \text{LSB}(\lceil c/p \rceil)\)
- \(\lceil \cdot \rceil\) returns nearest integer
- \(\sum (sk_t \ast z_t) = c \{\sum (sk_t \ast y_t)\} = c(1/p) \mod 2\)
Implementation Technique

- Function nearest_int
- Function LSB
Gentry’s FHE scheme

- KeyGen(λ)
- Encrypt(pk, m)
- Decrypt(sk, m)
- Evaluate(pk, f, c₁, ..., cₜ)
- Recrypt(pk₂, Dₐ, sk₁, c₁)
Evaluate \((pk, f, c_1, \ldots, c_t)\)

- Takes in boolean function with only ANDs and XORs
- Replaces AND with multiplication
- Replaces XOR with addition
- Returns \(c ← f(c_1, \ldots, c_t)\)
Implementation Technique

- Each $c_i$ is of type `publicKey`.
- Technically, computes $c.p \leftarrow f(c_1.p, \ldots, c_t.p)$
- $c.y$ is computed as $c.y_i \leftarrow pk.y_i \ast c.p$
- An expression evaluator was developed using stacks
Expression Evaluator

Input

Expression E and array of values

E[values]

Replace variables with values

Convert from infix to postfix format

Evaluate expression using a stack

Output

Result

Ep
Gentry’s FHE scheme

- KeyGen(λ)
- Encrypt(pk, m)
- Decrypt(sk, m)
- Evaluate(pk, f, c₁, ..., cₜ)
- Recrypt(pk₂, D_ε, sk₁, c₁)
Recrypt\((pk_2, D, sk_1, c_1)\)

- \(D\) is the boolean expression for the decryption function
- \(sk_1\) is a vector of cipher-texts, where \(sk_1[i] \leftarrow \text{Encrypt}(pk_2, sk_1[i])\)
- \(c_1\) is a cipher-text encrypted under \(pk_1\)
- Compute \(c_1\) : \(c_1[i] \leftarrow \text{Encrypt}(pk_2, \langle c_1 \rangle_i)\)
- Return \(c \leftarrow \text{Evaluate}(pk_2, D, sk_1, c_1)\)
Implementation Issues

- Formulation of D using naïve integer methods
- Published implementations till date [Gentry’ 11], [Smart’ 09] have used lattice based methods
## Timing Measurements

<table>
<thead>
<tr>
<th>Dimension</th>
<th>KeyGen</th>
<th>Encrypt</th>
<th>Decrypt</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^3$</td>
<td>0.405 ms</td>
<td>0.145 ms</td>
<td>0.125 ms</td>
</tr>
<tr>
<td>$2^5$</td>
<td>0.421 ms</td>
<td>0.337 ms</td>
<td>3.43 ms</td>
</tr>
<tr>
<td>$2^7$</td>
<td>0.422 ms</td>
<td>4.2 ms</td>
<td>16.36 ms</td>
</tr>
<tr>
<td>$2^9$</td>
<td>0.438 ms</td>
<td>33.37 ms</td>
<td>24.54 ms</td>
</tr>
<tr>
<td>$2^{11}$</td>
<td>0.437 ms</td>
<td>187.02 ms</td>
<td>89.16 ms</td>
</tr>
<tr>
<td>$2^{13}$</td>
<td>0.434 ms</td>
<td>474.29 ms</td>
<td>215.94 ms</td>
</tr>
<tr>
<td>$2^{15}$</td>
<td>0.433 ms</td>
<td>0.99 sec</td>
<td>0.5 sec</td>
</tr>
</tbody>
</table>
Short Term Goals

- Generalization of input by writing a convertor for boolean functions to AND-XOR form
- Use lattice-based methods to implement Recrypt
- Extensive testing
Long Term Goals

- Improve time and memory complexity of scheme. Current implementations are not practical
- Explore the possibilities of side-channel attacks on this scheme