Practical

Somewhat Homomorphic Encryption

David Wu
(joint work with Dan Boneh)
Homomorphic Encryption

Homomorphic encryption scheme: encryption scheme that allows computation on ciphertexts

Comprises of three functions:

Must satisfy usual notion of semantic security
Homomorphic Encryption

Homomorphic encryption scheme: encryption scheme that allows computation on ciphertexts

Comprises of three functions:

\[ c_1 = \text{Enc}_{pk}(m_1) \]
\[ c_2 = \text{Enc}_{pk}(m_2) \]
\[ \text{Dec}_{sk}\left(\text{Eval}_f(e_k, c_1, c_2)\right) = f(m_1, m_2) \]
Fully Homomorphic Encryption (FHE)

Many homomorphic encryption schemes:
- ElGamal: \( f(m_0, m_1) = m_0 m_1 \)
- Paillier: \( f(m_0, m_1) = m_0 + m_1 \)
- Goldwasser-Micali: \( f(m_0, m_1) = m_0 \oplus m_1 \)

Fully homomorphic encryption: homomorphic with respect to two operations: addition and multiplication
- Can evaluate Boolean and arithmetic circuits
- [BGN05]: one multiplication, many additions
- [Gen09]: first FHE construction from lattices
Fully Homomorphic Encryption

\[ c_1 = \text{Enc}_{pk}(m_1) \]
\[ c_2 = \text{Enc}_{pk}(m_2) \]

\[ \text{Eval} \]

\[ c_3 = \text{Eval}_f(ek, c_1, c_2) \]

\( C(f) \): circuit for some function \( f \)

**Correctness:** \( \text{Dec}_{sk}\left(\text{Eval}_f(ek, c_1, c_2)\right) = f(m_1, m_2) \)

**Circuit Privacy:** \( \text{Enc}_{pk}(C(m_1, m_2)) \approx \text{Eval}_f(ek, c_1, c_2) \)

**Compactness:** Decryption circuit has size at most \( \text{poly}(\lambda) \)
In Theory: Secure Computation using FHE

Represent “Search” function as a circuit and evaluate homomorphically
In Practice: Secure Computation using FHE

Enc(pk, "cat")

Request timed out

FHE schemes have tremendous overhead
Somewhat Homomorphic Encryption (SWHE)

FHE supports arbitrary number of operations
Compromise: Support a *limited* number of operations (e.g., evaluate circuits of a certain depth)
  • Somewhat/leveled homomorphic encryption
Brakerski’s SWHE [Bra12]

Operates over a polynomial ring: \( R = \mathbb{Z}[x]/\Phi_m(x) \)

Plaintext and ciphertext are vectors of ring elements

Homomorphic multiplication much more expensive than homomorphic addition
  • Can evaluate low degree polynomials over encrypted data
Application: Statistical Analysis

Local hospitals submit encrypted patient data

Cloud aggregates and summarizes patient data

Medical researcher investigating a disease outbreak

Model
Security Model

- Cloud does not learn patient data or model
- Researcher does not learn individual patient data other than what is explicitly leaked by model

(Circuit Privacy)

(Semantic Security)
Application: Statistical Analysis

Given $n$ vectors $x_1, \ldots, x_n$ (e.g., patient profiles), define $X$ to be the matrix with rows $x_1, \ldots, x_n$

- Mean: $\mu = \frac{1}{n} \sum_{i=1}^{n} x_i$
- Covariance: $\Sigma_X = \frac{1}{n^2} (nX^TX - (n\mu)(n\mu)^T)$

Division difficult to support, so represent as rationals

Depth 0 circuit for mean, depth 1 for covariance
Application: Statistical Analysis

Can also perform linear regression on encrypted data

Given design matrix $X$ and response vector $y$, evaluate normal equations:

$$ \theta = (X^T X)^{-1} X^T y $$

Invert over $\mathbb{Q}$ using Cramer’s rule

Depth $n$ for $n$ dimensional data
Batch Computation [SV11]

Encrypt + process array of values at no extra cost

Main intuition: Chinese Remainder Theorem

Plaintext Space: \( R_p = \mathbb{Z}_p[x]/\Phi_m(x) \)

Choose \( p \) such that \( R_p \) splits into smaller rings: \( R_p \approx \bigotimes_{i=1}^{n} R_{p_i} \)
Batch Computation

Encrypt + process array of values at no extra cost:

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
\hline
7 & 5 & 3 & 1 \\
\hline
8 & 7 & 6 & 5 \\
\end{array}
\]

In practice: $\geq 5000$ slots
Batch Computation

Can also permute slots (via Frobenius automorphisms) [BGV12, GHS12]
Batch Inner Products

Statistical analysis reduces to computing inner products:

\[
\begin{pmatrix}
1 \\
2 \\
3 \\
4
\end{pmatrix}^T \cdot 
\begin{pmatrix}
1 \\
0 \\
0 \\
1
\end{pmatrix} = 1 \cdot 1 + 2 \cdot 0 + 3 \cdot 0 + 4 \cdot 1 = 5
\]

Naïve method: Encrypt each component separately.

Requires 4 multiplications!
Batch Inner Products

Batch inner product: encrypt multiple components in each ciphertext.

Requires 1 multiplication!

Not quite what we wanted!
Batch Inner Products

Result of batch multiplication:

\[
\begin{array}{c}
1 \\
0 \\
0 \\
4 \\
\end{array}
\]

Desired result:

\[
5
\]
Batch Inner Products

Use automorphisms to sum up components:

\[
\begin{align*}
\nu_1 & \rightarrow \begin{bmatrix} 1 \\ 0 \\ 0 \\ 4 \end{bmatrix} & \rightarrow & \begin{bmatrix} 0 \\ 0 \\ 4 \\ 1 \end{bmatrix} & + & \begin{bmatrix} 1 \\ 0 \\ 0 \\ 4 \end{bmatrix} & \rightarrow & \begin{bmatrix} 1 \\ 0 \\ 4 \\ 5 \end{bmatrix} \\
\nu_2 & \rightarrow & \begin{bmatrix} 0 \\ 0 \\ 4 \\ 1 \end{bmatrix} & & & & \\
\nu_3 & \rightarrow & \begin{bmatrix} 0 \\ 0 \\ 4 \\ 1 \end{bmatrix} & & & & \\
\nu_4 & \rightarrow & \begin{bmatrix} 0 \\ 0 \\ 4 \\ 1 \end{bmatrix} & & & & \\
\end{align*}
\]

\(\nu_1 + \nu_2\)

\(\nu_3 + \nu_4\)
Batch Inner Products

Use automorphisms to sum up components:

\[
\begin{align*}
\nu_1 + \nu_2 & \quad \begin{bmatrix} 1 \\ 0 \\ 4 \\ 5 \end{bmatrix} & \quad \begin{bmatrix} 4 \\ 5 \\ 1 \\ 0 \end{bmatrix} \\
\nu_3 + \nu_4 & \quad \begin{bmatrix} 1 \\ 0 \\ 4 \\ 5 \end{bmatrix} & \quad \begin{bmatrix} 4 \\ 5 \\ 1 \\ 0 \end{bmatrix} \\
& \quad \begin{bmatrix} 5 \\ 5 \\ 5 \end{bmatrix} \\
& \quad \Sigma_i \nu_i
\end{align*}
\]
Batch Inner Products

$n$ multiplications

1 multiplication

$\log n$ automorphisms

$\log n$ additions

Batches of size $n$
Time to Compute Mean and Covariance over Encrypted Data (Dimension 4)

- Automorphisms dominate
- Multiplications dominate
Time to Compute Mean and Covariance over Encrypted Data (4096 Data Points)

Runtime grows as $O(d^2)$
Time to Perform Linear Regression on Encrypted Data (2 Dimensions)

- Automorphisms dominate
- Multiplications dominate
Time to Perform Linear Regression on Encrypted Data (260,000 Data Points)

Runtime grows as $O(d!)$.
Conclusions

SWHE allows computation of circuits of low-depth

Batching enables scaling to nontrivial datasets

Can perform statistical analysis on encrypted data with “reasonable” overhead
Open Source FHE Implementation:

https://github.com/dwu4/fhe-si