Private Decision Trees

CS 261 Fall 2018

Ryan Lehmkuhl
Fully Homomorphic Encryption
Fully Homomorphic Encryption

- Idea first proposed in 1978 by Rivest et al
Fully Homomorphic Encryption

- Idea first proposed in 1978 by Rivest et al
- Given:
  
  $Enc(m), f, pk$
Fully Homomorphic Encryption

- Idea first proposed in 1978 by Rivest et al
- Given:
  \[ \text{Enc}(m), f, \text{pk} \]
- Some function \( \text{Evaluate} \) exists such that:
  \[ \text{Evaluate}(pk, f, \text{Enc}(m)) \longrightarrow \text{Enc}(f(m)) \]
Fully Homomorphic Encryption

- Idea first proposed in 1978 by Rivest et al
- Given:
  \[ Enc(m), f, pk \]
- Some function \textit{Evaluate} exists such that:
  \[ \text{Evaluate}(pk, f, Enc(m)) \rightarrow Enc(f(m)) \]
- First construction in 2009 by Craig Gentry using ideal lattices (won ACM best doctoral dissertation award)
Fully Homomorphic Encryption

Pros:
Fully Homomorphic Encryption

Pros:

- IT’S REALLY POWERFUL
Fully Homomorphic Encryption

Pros:

- IT’S REALLY POWERFUL

Cons:
Fully Homomorphic Encryption

Pros:

• IT’S REALLY POWERFUL

Cons:

• General mechanism → lots of overhead
Fully Homomorphic Encryption

Pros:

• IT’S REALLY POWERFUL

Cons:

• General mechanism $\rightarrow$ lots of overhead
• Malleability isn’t always desired (El Gamal, RSA)
Fully Homomorphic Encryption
Fully Homomorphic Encryption

- FHE is basically a hidden error-correcting code
Fully Homomorphic Encryption

- FHE is basically a hidden error-correcting code
- Every operation adds noise... at a certain point the plaintext can’t be recovered
Fully Homomorphic Encryption

- FHE is basically a hidden error-correcting code
- Every operation adds noise... at a certain point the plaintext can’t be recovered
- Bootstrappable encryption schemes solve this problem but at a high cost
Fully Homomorphic Encryption

- FHE is basically a hidden error-correcting code
- Every operation adds noise... at a certain point the plaintext can’t be recovered
- Bootstrappable encryption schemes solve this problem but at a high cost

What can we do?
Fully Homomorphic Encryption

- FHE is basically a hidden error-correcting code
- Every operation adds noise... at a certain point the plaintext can’t be recovered
- Bootstrappable encryption schemes solve this problem but at a high cost

What can we do?
  → Use a leveled FHE scheme!
Fully Homomorphic Encryption

- FHE is basically a hidden error-correcting code
- Every operation adds noise... at a certain point the plaintext can’t be recovered
- Bootstrappable encryption schemes solve this problem but at a high cost

What can we do?
   → Use a leveled FHE scheme!

- Tends to be more efficient than complete FHE, especially for circuits with a small number of levels
FHE SIMD Slots
FHE SIMD Slots

- SIMD = Single Instruction Multiple Data
FHE SIMD Slots

- **SIMD = Single Instruction Multiple Data**
- Used in hardware to allow parallel execution of a single instruction on multiple registers
FHE SIMD Slots

- SIMD = Single Instruction Multiple Data
- Used in hardware to allow parallel execution of a single instruction on multiple registers
- FHE schemes exist which support packing multiple messages into a single ciphertext - called *SIMD Slots*
FHE SIMD Slots

- **SIMD** = Single Instruction Multiple Data
- Used in hardware to allow parallel execution of a single instruction on multiple registers
- FHE schemes exist which support packing multiple messages into a single ciphertext - called *SIMD Slots*

\[ c \leftarrow Enc(m_1, m_2, \ldots, m_n) \]
FHE SIMD Slots

- SIMD = Single Instruction Multiple Data
- Used in hardware to allow parallel execution of a single instruction on multiple registers
- FHE schemes exist which support packing multiple messages into a single ciphertext - called *SIMD Slots*

\[
c \leftarrow \text{Enc}(m_1, m_2, \ldots, m_n)
\]

\[
\text{Evaluate}(f, c) \rightarrow \text{Enc}(f(m_1), f(m_2), \ldots, f(m_n))
\]
Fully Homomorphic Encryption

Open-source libraries exist for doing FHE which include many more optimizations! (eg. HE Lib)
Fully Homomorphic Encryption

Open-source libraries exist for doing FHE which include many more optimizations! (eg. HE Lib)

In summary we’ll try to:
Fully Homomorphic Encryption

Open-source libraries exist for doing FHE which include many more optimizations! (eg. HE Lib)

In summary we’ll try to:

→ Minimize the depth of our function circuit
Fully Homomorphic Encryption

Open-source libraries exist for doing FHE which include many more optimizations! (eg. HE Lib)

In summary we’ll try to:
→ Minimize the depth of our function circuit
→ Use FHE SIMD Slots to parallelize repeated instructions
Fully Homomorphic Encryption

Open-source libraries exist for doing FHE which include many more optimizations! (eg. HE Lib)

In summary we’ll try to:

→ Minimize the depth of our function circuit
→ Use FHE SIMD Slots to parallelize repeated instructions
→ Work in $\mathbb{F}_2$
Decision Tree Classifiers
Decision Tree Classifiers

• Partitions the feature vector space on each attribute
Decision Tree Classifiers

- Partitions the feature vector space on each attribute
- Interior nodes correspond to partitioning rules
Decision Tree Classifiers

- Partitions the feature vector space on each attribute
- Interior nodes correspond to partitioning rules
- Leaf nodes correspond to class labels
Secure Decision Trees
Secure Decision Trees

- Represent tree as a polynomial function $P$
Secure Decision Trees

- Represent tree as a polynomial function $P$
- Perform multiplications in a tree-like manner to reduce the depth of the polynomial circuit from $h_{max}$ to $\log_2 h_{max}$
Secure Decision Trees

- Represent tree as a polynomial function $P$
- Perform multiplications in a tree-like manner to reduce the depth of the polynomial circuit from $h_{\text{max}}$ to $\log_2 h_{\text{max}}$
- In order to use $\mathbb{F}_2$ for FHE speedup, we encode each class as a binary string and use SIMD slots to parallelize