KU LEUVEN

Revisiting Oblivious Top-k Selection with Applications to Secure k-NN Classification

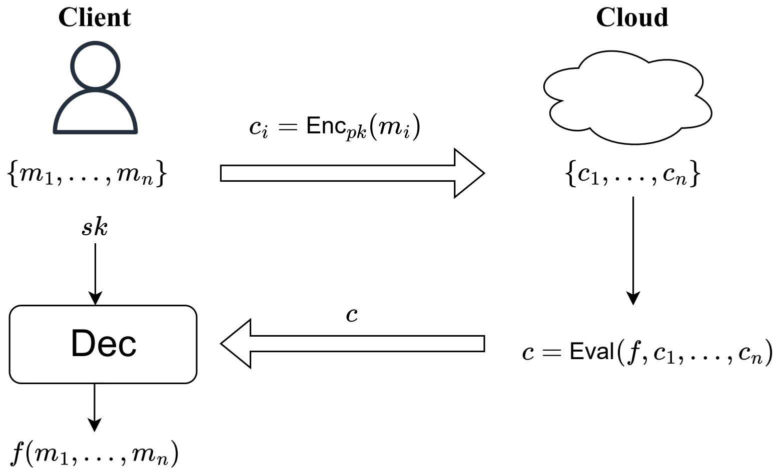
Kelong Cong[1,2], Robin Geelen[1], **Jiayi Kang**[1], and Jeongeun Park[1]

[1]COSIC, KU Leuven, and [2]Zama

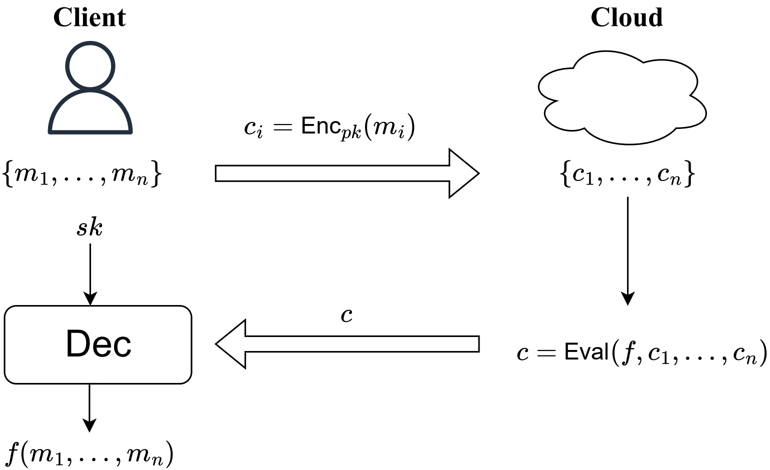
Seminar at University of Luxembourg, March 14, 2024

Outline

FHE supports secure computation outsourcing



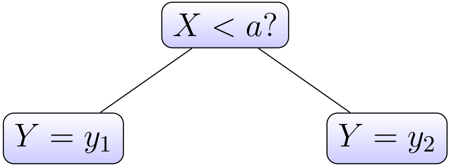
FHE supports secure computation outsourcing



- Promising future: imagine asking ChatGPT encrypted questions!

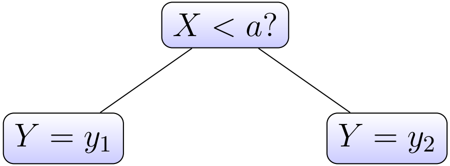
Program expansion in homomorphic branching

- Converting input-dependent plaintext programs into ciphertext programs leads to program expansion
- Example of program expansion:



Program expansion in homomorphic branching

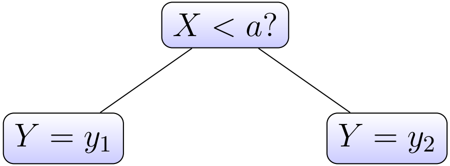
- Converting input-dependent plaintext programs into ciphertext programs leads to program expansion
- Example of program expansion:



1. Homomorphically compute branch $b = \mathbb{1}(X < a)$

Program expansion in homomorphic branching

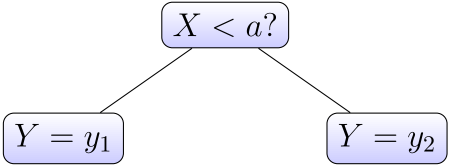
- Converting input-dependent plaintext programs into ciphertext programs leads to program expansion
- Example of program expansion:



1. Homomorphically compute branch $b = \mathbb{1}(X < a)$
2. Homomorphically evaluate $Y = (1 - b) \cdot y_1 + b \cdot y_2$

Program expansion in homomorphic branching

- Converting input-dependent plaintext programs into ciphertext programs leads to program expansion
- Example of program expansion:



1. Homomorphically compute branch $b = \mathbb{1}(X < a)$
2. Homomorphically evaluate $Y = (1 - b) \cdot y_1 + b \cdot y_2$

- *Both* child nodes need to be visited

Oblivious programs and their network realization

Definition

(Data-)oblivious programs are programs whose sequence of operations and memory accesses are independent of inputs.

KU LEUVEN

Oblivious programs and their network realization

Definition

(Data-)oblivious programs are programs whose sequence of operations and memory accesses are independent of inputs.

- Consider comparator-based sortings for d elements
  - Quicksort has complexity $\mathcal{O}(d \log d)$, but it is non-oblivious
  - Practical oblivious sorting method has complexity $\mathcal{O}(d \log^2 d)$

Oblivious programs and their network realization

Definition

(Data-)oblivious programs are programs whose sequence of operations and memory accesses are independent of inputs.

- Consider comparator-based sortings for d elements
  - Quicksort has complexity $\mathcal{O}(d \log d)$, but it is non-oblivious
  - Practical oblivious sorting method has complexity $\mathcal{O}(d \log^2 d)$
- Oblivious programs can be visualized as networks

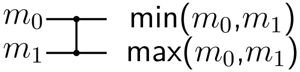


Figure: Comparator

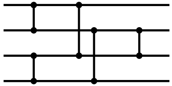
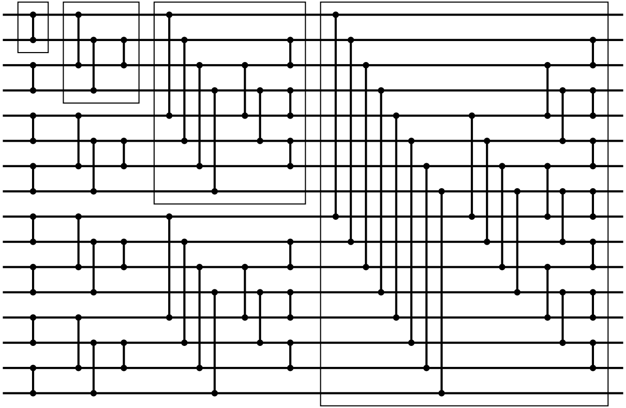


Figure: Sort 4 elements obliviously

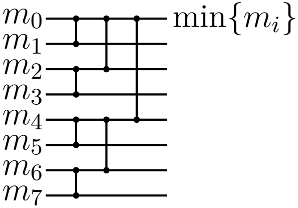
Example: Batcher's odd-even sorting network

- Batcher's odd-even sorting network for d input elements has complexity $S(d) = \mathcal{O}(d \log^2 d)$ and depth $\mathcal{O}(\log^2 d)$



Example: the tournament network for Min/Max

- The tournament network for d input elements has complexity $d-1$ and depth $\lceil \log d \rceil$



KU LEUVEN

Outline

1. Oblivious Algorithms for Secure Computation
2. Oblivious Top-k Selection
3. Application: Secure k-NN Classification
4. Summary and Conclusion

Motivation for Top-k selection problem

Definition

Given a set of d elements, a *Top-k algorithm* selects its k smallest (or largest) elements.

Motivation for Top-k selection problem

Definition

Given a set of d elements, a *Top-k algorithm* selects its k smallest (or largest) elements.

- In the huge information space (consisting of d records), only k most important records are of interest:
  1. define a proper scoring function
  2. compute score of all d records
  3. return the k records with the highest scores

Motivation for Top-k selection problem

Definition

Given a set of d elements, a *Top-k algorithm* selects its k smallest (or largest) elements.

- In the huge information space (consisting of d records), only k most important records are of interest:
  1. define a proper scoring function
  2. compute score of all d records
  3. return the k records with the highest scores
- Example applications include
  - k-nearest neighbors classification
  - recommender systems
  - genetic algorithms

Popular oblivious Top-k methods

- The first category uses an oblivious sorting algorithm and then discards the $d - k$ irrelevant elements:

Popular oblivious Top-k methods

- The first category uses an oblivious sorting algorithm and then discards the $d - k$ irrelevant elements:
  - Batcher's odd-even merge sort with complexity $\mathcal{O}(d \log^2 d)$ and depth $\mathcal{O}(\log^2 d)$

Popular oblivious Top-k methods

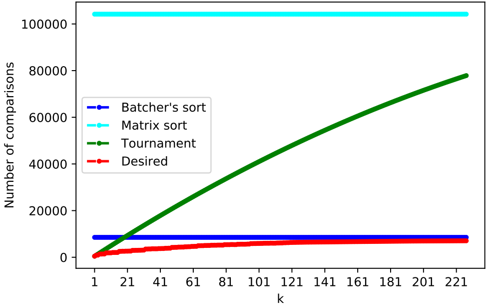
- The first category uses an oblivious sorting algorithm and then discards the $d - k$ irrelevant elements:
  - Batcher's odd-even merge sort with complexity $\mathcal{O}(d \log^2 d)$ and depth $\mathcal{O}(\log^2 d)$
  - Comparison matrix method with complexity $\mathcal{O}(d^2)$ and constant depth

Popular oblivious Top-k methods

- The first category uses an oblivious sorting algorithm and then discards the $d - k$ irrelevant elements:
  - Batcher's odd-even merge sort with complexity $\mathcal{O}(d \log^2 d)$ and depth $\mathcal{O}(\log^2 d)$
  - Comparison matrix method with complexity $\mathcal{O}(d^2)$ and constant depth
- The second category obliviously computes the minimum k times
  - Complexity $\mathcal{O}(kd)$ and depth $\mathcal{O}(k \log d)$

Popular oblivious Top-k methods

- The first category uses an oblivious sorting algorithm and then discards the $d - k$ irrelevant elements:
  - Batcher's odd-even merge sort with complexity $\mathcal{O}(d \log^2 d)$ and depth $\mathcal{O}(\log^2 d)$
  - Comparison matrix method with complexity $\mathcal{O}(d^2)$ and constant depth
- The second category obliviously computes the minimum k times
  - Complexity $\mathcal{O}(kd)$ and depth $\mathcal{O}(k \log d)$



Alekseev's oblivious Top-k for $2k$ elements

- Realization using two building blocks:
  - Sorting network of size k
  - Pairwise comparison: returns the Top-k elements

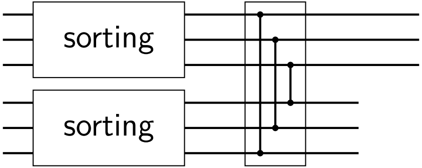


Figure: Example for $k = 3$

Alekseev's oblivious Top-k for $2k$ elements

- Realization using two building blocks:
  - Sorting network of size k
  - Pairwise comparison: returns the Top-k elements

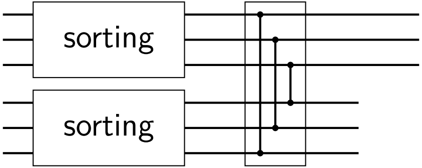
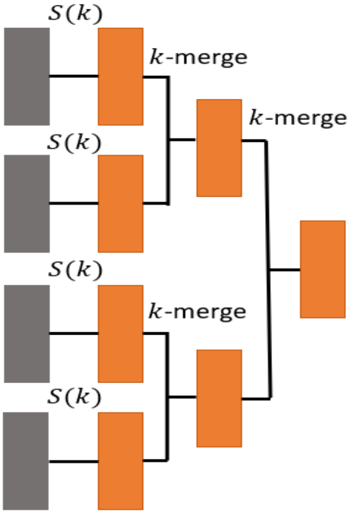


Figure: Example for $k = 3$

- Can be generalized to Top-k out of d elements in tournament manner

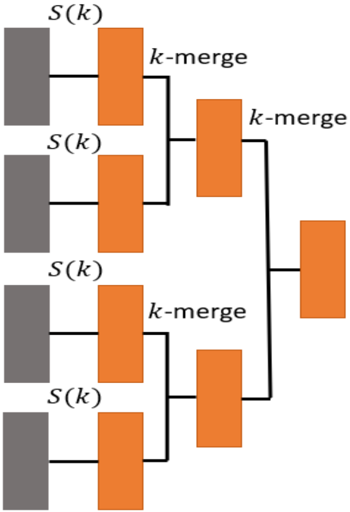
Alekseev's oblivious Top-k for d elements



- Alekseev's procedure realizes k-merge as pairwise comparison followed by sorting

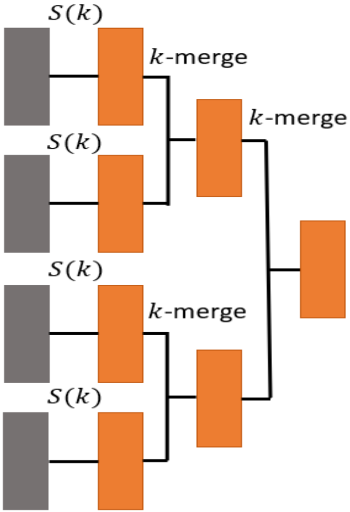
KU LEUVEN

Alekseev's oblivious Top-k for d elements



- Alekseev's procedure realizes k-merge as pairwise comparison followed by sorting
- Complexity of k-merge is $k + S(k)$ comparators

Alekseev's oblivious Top-k for d elements



- Alekseev's procedure realizes k-merge as pairwise comparison followed by sorting
- Complexity of k-merge is $k + S(k)$ comparators
- Alekseev's Top-k for d elements has complexity

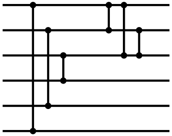
$$\mathcal{O}(d \log^2 k),$$

assuming practical $S(k) = \mathcal{O}(k \log^2 k)$

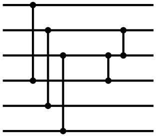
KU LEUVEN

Improvement I: order-preserving merge

- Batcher's odd-even sorting network uses an alternative merging approach
  - We realize k-merge by removing redundant comparators in Batcher's merge
  - This reduces the complexity from $\mathcal{O}(k \log^2 k)$ in Alekseev's k-merge to $\mathcal{O}(k \log k)$



(a) Alekseev's 3-merge



(b) Our 3-merge

Improvement I: oblivious Top-k from truncation

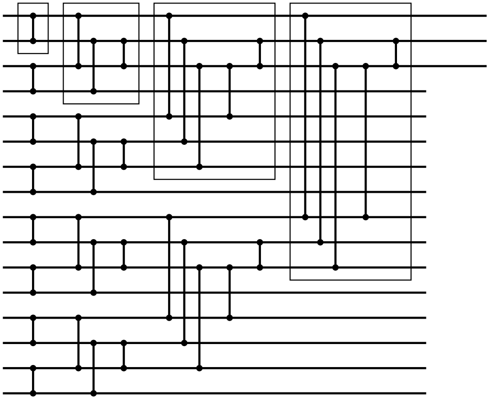
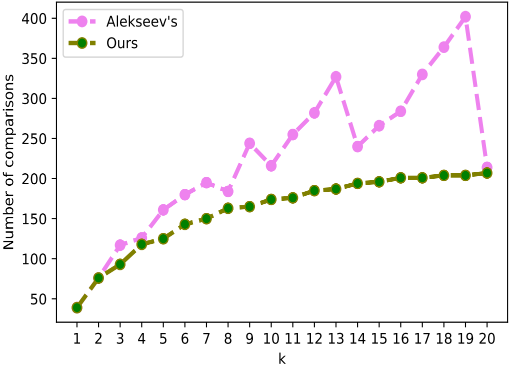


Figure: Our truncated sorting network for finding the 3 smallest values out of 16

KU LEUVEN

Improvement I: comparison

- Our Top-k method for d elements has the same asymptotic complexity as Alekseev's: $\mathcal{O}(d \log^2 k)$ comparators
- Our solution contains fewer comparators in practice



KU LEUVEN

Revisiting Yao's oblivious Top-k

- Andrew Yao improved Alekseev's Top-k using an unbalanced recursion

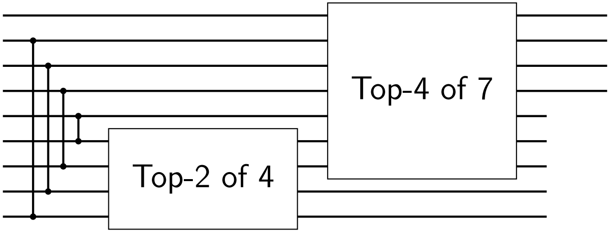


Figure: Selecting Top-4 of 9 elements using Yao's method

Revisiting Yao's oblivious Top-k

- Andrew Yao improved Alekseev's Top-k using an unbalanced recursion

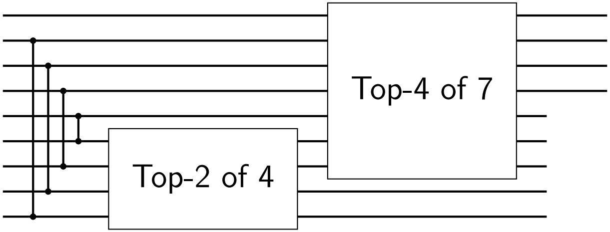


Figure: Selecting Top-4 of 9 elements using Yao's method

- For $k \ll \sqrt{d}$, Yao's Top-k method has complexity $\mathcal{O}(d \log k)$

Revisiting Yao's oblivious Top-k

- Andrew Yao improved Alekseev's Top-k using an unbalanced recursion

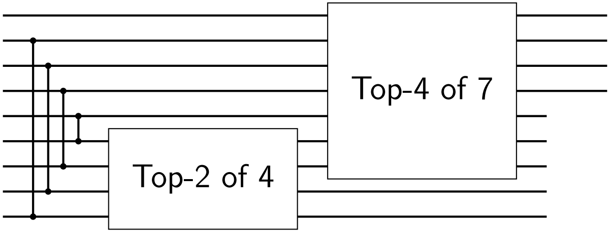
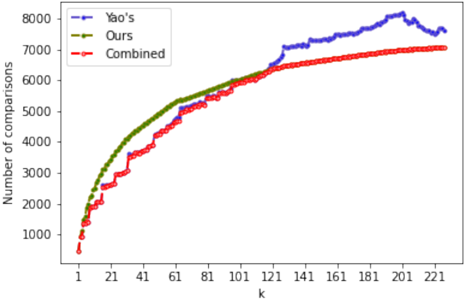


Figure: Selecting Top-4 of 9 elements using Yao's method

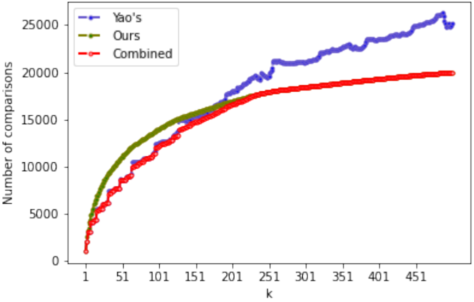
- For $k \ll \sqrt{d}$, Yao's Top-k method has complexity $\mathcal{O}(d \log k)$
- For $k \gg \sqrt{d}$, the complexity of Yao's Top-k method is asymptotically higher than $\mathcal{O}(d \log^2 k)$

Improvement II: combining our method with Yao's

- The combined network recursively calls our truncation method or Yao's method, depending on which one uses fewer comparators



(a) $d = 457$



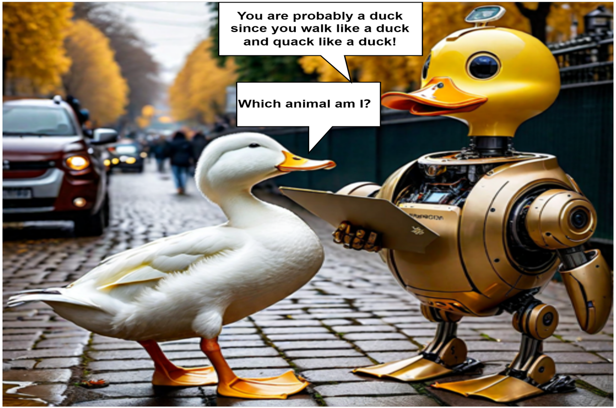
(b) $d = 1000$

KU LEUVEN

Outline

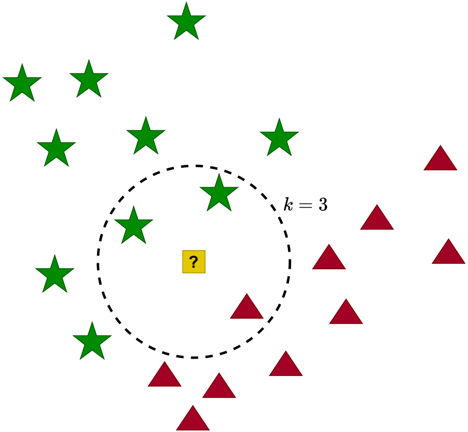
Introduction to k-Nearest Neighbors (k-NN)

- Simple machine learning algorithm with broad applications
  - Web and image search, plagiarism detection, sports player recruitment, ...



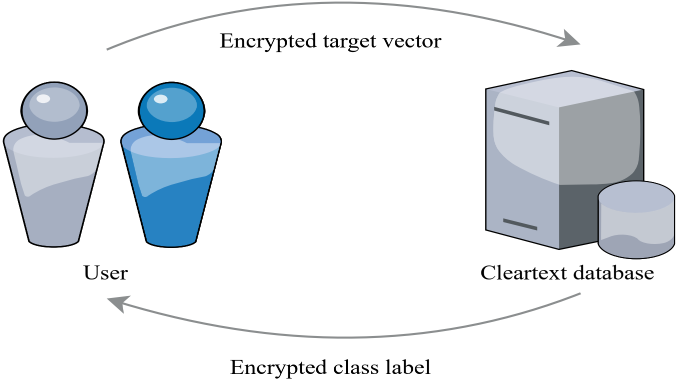
Introduction to k-Nearest Neighbors (k-NN)

- Three-step method:
  1. Compute distance between target vector and d database vectors
  2. Find k closest database vectors and corresponding labels
  3. Class assignment is majority vote of these k labels



Secure k-NN threat model

- Client sends encrypted k-NN query to server
- Server returns encrypted classification result

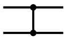


KU LEUVEN

Homomorphic realization of k-NN

1. Compute distance between target vector and d database vectors
   - Relatively cheap
2. Find k closest database vectors and corresponding labels
   - Top-k network built from comparators
   - Each comparator is realized with two bootstrappings
     - One bootstrapping for the minimum and maximum
     - One bootstrapping for the corresponding class labels

$$(\text{dist}_0, \text{label}_0) \qquad (\text{dist}_i, \text{label}_i)$$



$$(\text{dist}_1, \text{label}_1) \qquad (\text{dist}_{1-i}, \text{label}_{1-i})$$

   - Where $i = \arg\min(\text{dist}_0, \text{dist}_1)$
3. Class assignment is majority vote of these k labels

Performance for MNIST dataset

- Implementation in tfhe-rs

k	d	Comparators [ZS21]	Comparators Ours	Duration (s) [ZS21]	Duration (s) Ours	Duration (s) Speedup
3	40	780	93	30	17.5	1.7×
	457	104196	1136	4248	202.3	21×
	1000	499500	2493	20837	441.1	47.2×
$\lfloor\sqrt{d}\rfloor$	40	780	143	33	28.1	1.2×
	457	104196	3412	4402	530.2	8.3×
	1000	499500	9121	21410	1252	17.1×

Outline

Conclusion

- An oblivious Top-k algorithm that has complexity
  - $\mathcal{O}(d \log^2 k)$ in general
  - $\mathcal{O}(d \log k)$ for small $k \ll \sqrt{d}$
- Top-k is an important submodule for various applications
  - For secure k-NN, the Top-k network leads to $47\times$ speedup compared to [ZS21]

Thank you for your attention!

ia.cr/2023/852