Kelong Cong[1], Robin Geelen[2], Jiayi Kang[2], and Jeongeun Park[3]
[1]Zama ▪ [2]COSIC, KU Leuven ▪ [3]NTNU

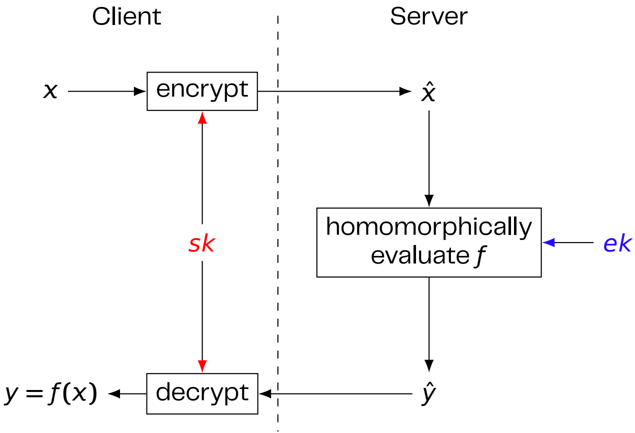
28 Aug 2024

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

SAC 2024

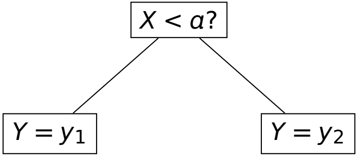
ZAMA

FHE supports secure computation outsourcing



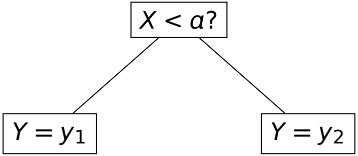
Program expansion in homomorphic branching

- **Program expansion** happens when converting input-dependent plaintext programs into ciphertext programs
- Example of program expansion:



Program expansion in homomorphic branching

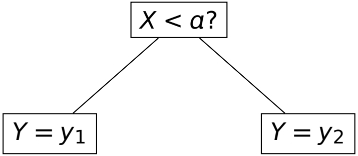
- **Program expansion** happens when converting input-dependent plaintext programs into ciphertext programs
- Example of program expansion:



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

- **Program expansion** happens when converting input-dependent plaintext programs into ciphertext programs
- 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 algorithms whose sequence of operations and memory accesses are independent of inputs.

Oblivious programs and their network realization

Definition

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

- Example: sorting d elements
  - Quicksort has complexity $\mathcal{O}(d \log d)$ but is not oblivious
  - Practical oblivious sorting methods have complexity $\mathcal{O}(d \log^2 d)$

Oblivious programs and their network realization

Definition

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

- Example: sorting d elements
  - Quicksort has complexity $\mathcal{O}(d \log d)$ but is not oblivious
  - Practical oblivious sorting methods have complexity $\mathcal{O}(d \log^2 d)$
- Oblivious programs are visualized as **networks**

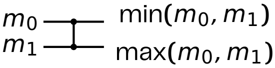


Figure: Comparator

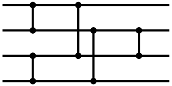
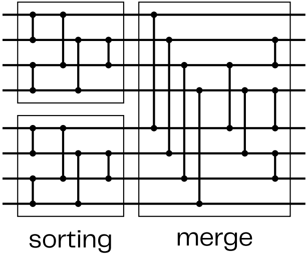


Figure: Sorting 4 elements obliviously

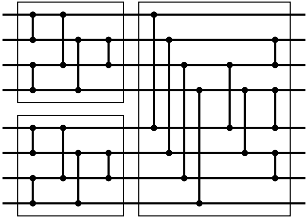
Example: Batcher's odd-even sorting network

- Built from recursive sortings followed by merge



Example: Batcher's odd-even sorting network

- Built from recursive sortings followed by merge



- Batcher's odd-even sorting network has
  - Complexity $S(d) = \mathcal{O}(d \log^2 d)$
  - Depth $\mathcal{O}(\log^2 d)$

Motivation for Top-k selection problem

Definition

A *Top-k algorithm* selects the k smallest elements from an array of d elements.

- In huge information space, 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

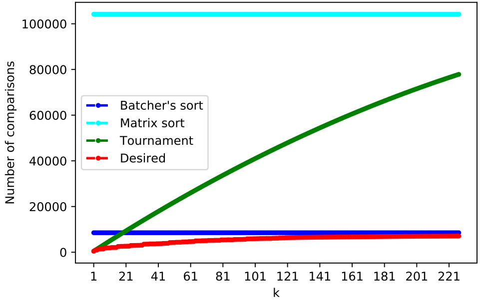
Definition

A *Top-k algorithm* selects the k smallest elements from an array of d elements.

- In huge information space, 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

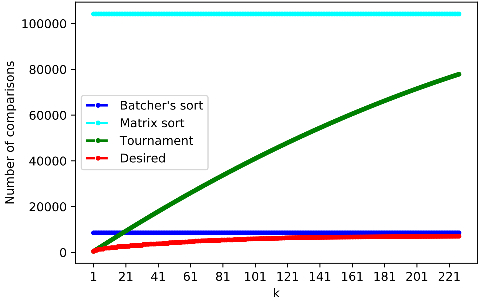
Oblivious Top-k methods from prior work

- First category: oblivious sorting, then discard $d - k$ 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 with complexity $\mathcal{O}(d^2)$ and constant depth



Oblivious Top-*k* methods from prior work

- First category: oblivious sorting, then discard $d - k$ 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 with complexity $\mathcal{O}(d^2)$ and constant depth
- Second category: compute minimum k times
  - Complexity $\mathcal{O}(kd)$ and depth $\mathcal{O}(k \log d)$

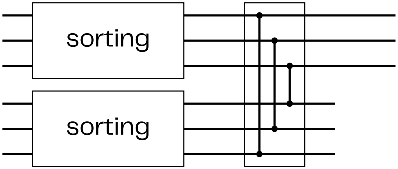


Our contribution

- Revisit classical oblivious Top-k selection methods (Alekseev '69 and Yao '80)
- Build upon them to build a network that performs well in all scenarios
- Apply the network in k-NN

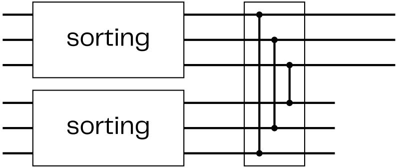
Alekseev's oblivious Top-*k* for 2*k* elements

- Realization using two building blocks:
  - Sorting network of size *k*
  - Pairwise comparison



Alekseev's oblivious Top-k for 2k elements

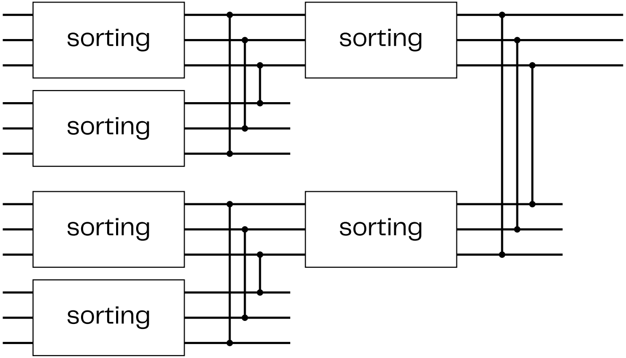
- Realization using two building blocks:
  - Sorting network of size k
  - Pairwise comparison



- Can be generalized to Top-k out of d elements

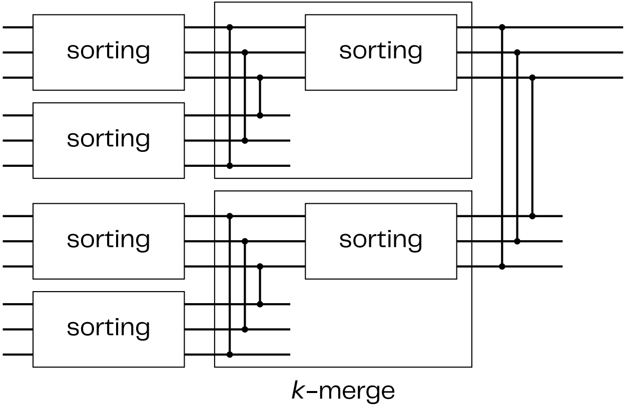
Alekseev's oblivious Top-*k* for *d* elements

- Top-*k* complexity is $\mathcal{O}(d \log^2 k)$ if $S(k) = \mathcal{O}(k \log^2 k)$



Alekseev's oblivious Top-k for d elements

- Top-k complexity is $\mathcal{O}(d \log^2 k)$ if $S(k) = \mathcal{O}(k \log^2 k)$



- Realizes k-merge as pairwise comparison + sorting: complexity $k + S(k)$

Improvement I: order-preserving merge

- Batcher's odd-even sorting network uses alternative merge
  - Truncate to k-merge by removing redundant comparators
  - Complexity reduction from $\mathcal{O}(k \log^2 k)$ to $\mathcal{O}(k \log k)$

(a) Alekseev's 3-merge **(b)** Our 3-merge

Improvement I: oblivious Top-*k* from truncation

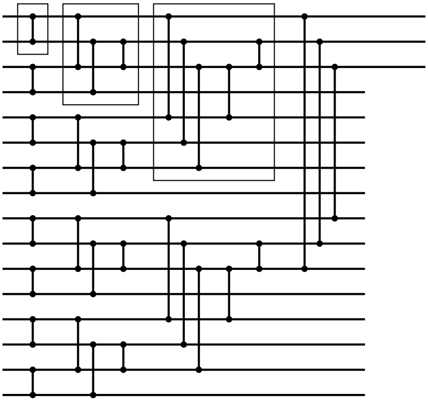
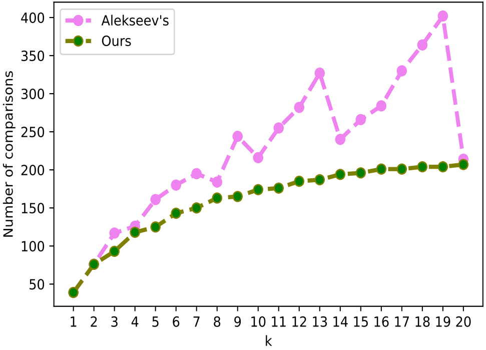


Figure: Network realization for Top-3 of 16 elements

Improvement I: comparison

- Same asymptotic complexity as Alekseev: $\mathcal{O}(d \log^2 k)$ comparators
- Our solution contains fewer comparators in practice



Revisiting Yao's oblivious Top-*k*

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

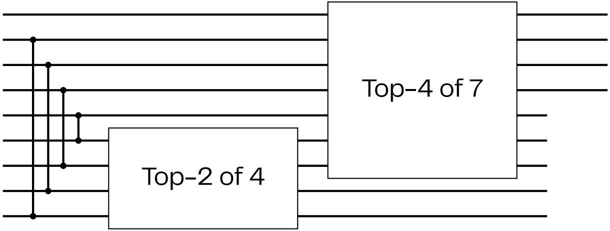


Figure: Selecting Top-4 of 9 elements using Yao's method

Revisiting Yao's oblivious Top-k

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

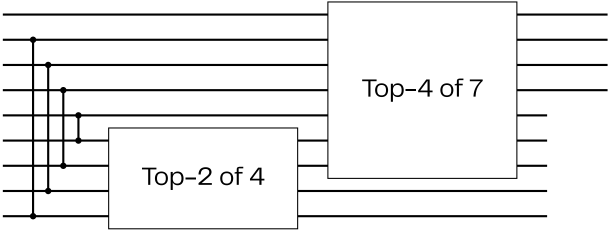
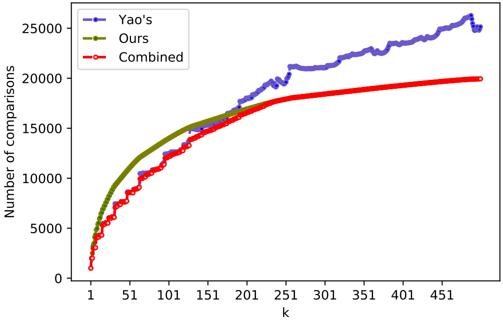


Figure: Selecting Top–4 of 9 elements using Yao's method

- $k \ll \sqrt{d}$: complexity is $\mathcal{O}(d \log k)$, better than before
- $k \gg \sqrt{d}$: complexity is asymptotically higher than $\mathcal{O}(d \log^2 k)$

Improvement II: combining our method with Yao’s

- Combined network recursively calls our or Yao’s method
- Slightly improves on the better method in some cases

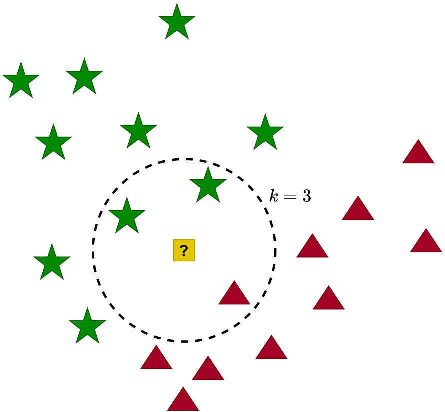


Introduction to *k*-Nearest Neighbors

- Simple machine learning algorithm with broad applications
  - Plagiarism detection, image classification, intrusion detection, ...
  - Lazy learning: no training phase

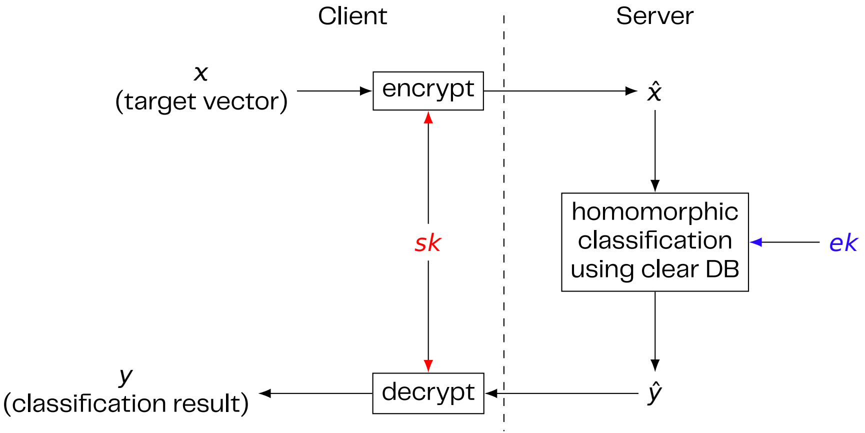
Introduction to *k*-Nearest Neighbors

- 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

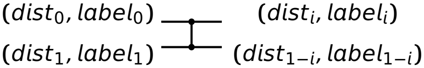


Homomorphic realization of k-NN

1. Compute distance between target vector and d database vectors

Homomorphic realization of k-NN

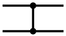
1. Compute distance between target vector and d database vectors
2. Find k closest database vectors and corresponding labels
   - Top-k network is built from comparators
   - Each comparator uses two programmable bootstrappings



using $i = \arg\min(dist_0, dist_1)$

Homomorphic realization of k-NN

1. Compute distance between target vector and d database vectors
2. Find k closest database vectors and corresponding labels
   - Top-k network is built from comparators
   - Each comparator uses two programmable bootstrappings

$(dist_0, label_0)$  $(dist_i, label_i)$

$(dist_1, label_1)$ $(dist_{1-i}, label_{1-i})$

using $i = \arg\min(dist_0, dist_1)$

3. Class assignment is majority vote of these k labels

Performance for MNIST dataset

- Implementation in tfhe-rs (`https://github.com/kuleuven-cosic/ppknn`)
- Difference for larger d is because [ZS21]† uses a $O(d^2)$ algorithm

k	d	Comparators [ZS21]†	Comparators Ours	Duration (s) [ZS21]†	Duration (s) Ours	Duration (s) Speedup
3	40	780	93	30	**18**	**1.7**×
	457	104196	1136	4248	**202**	**21.0**×
	1000	499500	2493	20837	**441**	**47.2**×
$\lfloor \sqrt{d} \rfloor$	40	780	143	33	**28**	**1.2**×
	457	104196	3412	4402	**530**	**8.3**×
	1000	499500	9121	21410	**1252**	**17.1**×

†Zuber and Sirdey: Efficient homomorphic evaluation of k-NN classifiers

Conclusion

- An oblivious Top-k algorithm with complexity
  - $\mathcal{O}(d \log^2 k)$ in general
  - $\mathcal{O}(d \log k)$ for small $k \ll \sqrt{d}$
  - By revisiting classical Top-k selection networks

Conclusion

- An oblivious Top-k algorithm with complexity
  - $\mathcal{O}(d \log^2 k)$ in general
  - $\mathcal{O}(d \log k)$ for small $k \ll \sqrt{d}$
  - By revisiting classical Top-k selection networks
- Implementation of a secure k-NN classifier in TFHE-rs
  - Feasible for database of 1000 records: $47\times$ faster than [ZS21]

Thank you.

ZAMA

Contact and Links

ia.cr/2023/852

github.com/kuleuven-cosic/ppknn

kelong.cong@zama.ai

ZAMA