UNIVERSITY OF OSLO

FFI Forsvarets forskningsinstitutt
Norwegian Defence Research Establishment

Master's Defense

Private Database Search

Benjamin Hansen Mortensen
University of Oslo
Norwegian Defence Research Establishment

14.06.2024

Agenda

1. Introduce
2. Demonstrate
3. Explain
4. Conclude

Introduction

PNR registry

- Chapter 60 of Politiregisterforskriften, FOR-2022-04-29-646, introduces the PNR registry
  - Collects information about itineraries and passengers from airlines
  - The registry can be used by other competent authorizes given sufficient justification
- The problem:
  - Apparent deadlock situation when the client’s query contains classified information
- Thesis statement:
  - It is feasible to use secure computation to enhance the privacy and capability of queries on the PNR registry compared to today’s disclose all approach, with respect to Politiregisterforskriften chapter 60, under the constraints of a two-party setting with communication over LAN and with security against semi-honest adversaries.

Goal

- Study MPC and relevant subfields to identify properties of a suitable solution:
  - Solution lies in the subfield of OT
- Desired functionality:
  - Keyword Search
  - Semantic Search
  - File Retrieval
- Some desired properties:
  - Access control
  - Adaptive
  - Timely

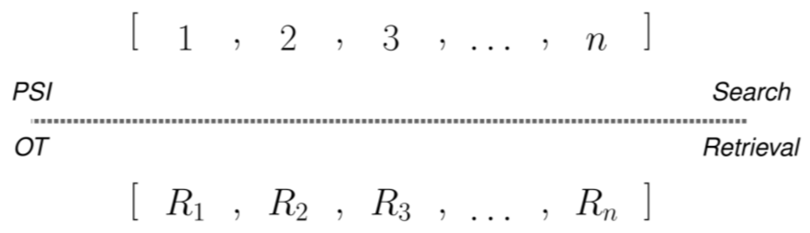
Demonstration

Explaination

Core Idea

Private Database Search

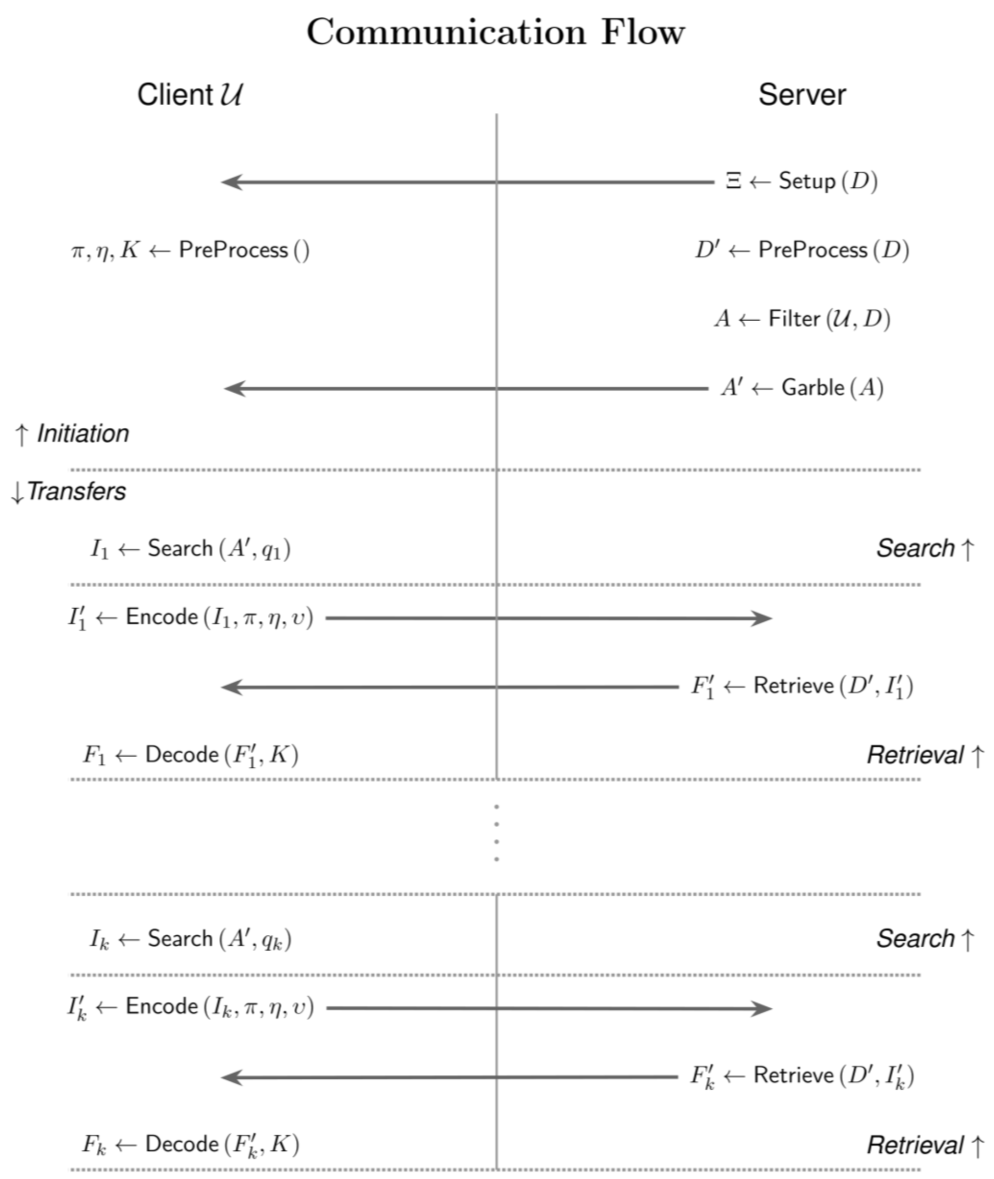
- Two parts:
  - Search is performed on the indices
  - Retrieval is performed on the records
- Important that the indices are independent of the records
  - The order of the database is therefore shuffled
- Allows us to combine different types of searches with various retrieval mechanisms



Communication Flow

Private Database Search

- Consists of seven main algorithms:
  - **Setup**: Configures the database
  - **PreProcess:** Encodes the database
  - **Filter**: Derives some auxiliary information about the database
  - **Garble**: Encodes the auxiliary information
  - **Search**: Searches the auxiliary information to find relevant record indices
  - **Encode**: Encodes the record indices
  - **Retrieve**: Retrieves the records for the indices
  - **Decode**: Decodes the records



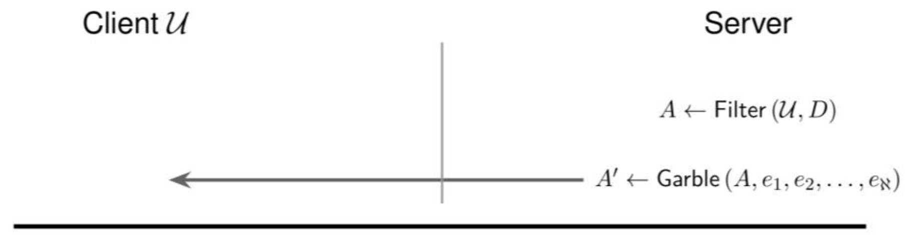
Communication Flow
Client $\mathcal{U}$
Server
$\Xi \leftarrow \text{Setup}(D)$
$\pi, \eta, K \leftarrow \text{PreProcess}()$
$D' \leftarrow \text{PreProcess}(D)$
$A \leftarrow \text{Filter}(\mathcal{U}, D)$
$A' \leftarrow \text{Garble}(A)$
↑ Initiation
↓ Transfers
$I_1 \leftarrow \text{Search}(A', q_1)$
Search ↑
$I'_1 \leftarrow \text{Encode}(I_1, \pi, \eta, \upsilon)$
$F'_1 \leftarrow \text{Retrieve}(D', I'_1)$
$F_1 \leftarrow \text{Decode}(F'_1, K)$
Retrieval ↑
$I_k \leftarrow \text{Search}(A', q_k)$
Search ↑
$I'_k \leftarrow \text{Encode}(I_k, \pi, \eta, \upsilon)$
$F'_k \leftarrow \text{Retrieve}(D', I'_k)$
$F_k \leftarrow \text{Decode}(F'_k, K)$
Retrieval ↑

Keyword Search

Initiation

- **Filter** creates an inverted index matrix of the database where information is filtered according to some policy.
  - The first column does not contain duplicates
- **Garble** encrypts the indexing using an ephemeral key for each column:
  - The first column uses the key directly
  - For the other columns each cell uses a unique key

Communication Flow



Client $\mathcal{U}$
Server
$A \leftarrow$ Filter $(\mathcal{U}, D)$
$A' \leftarrow$ Garble $(A, e_1, e_2, \dots, e_{\aleph})$

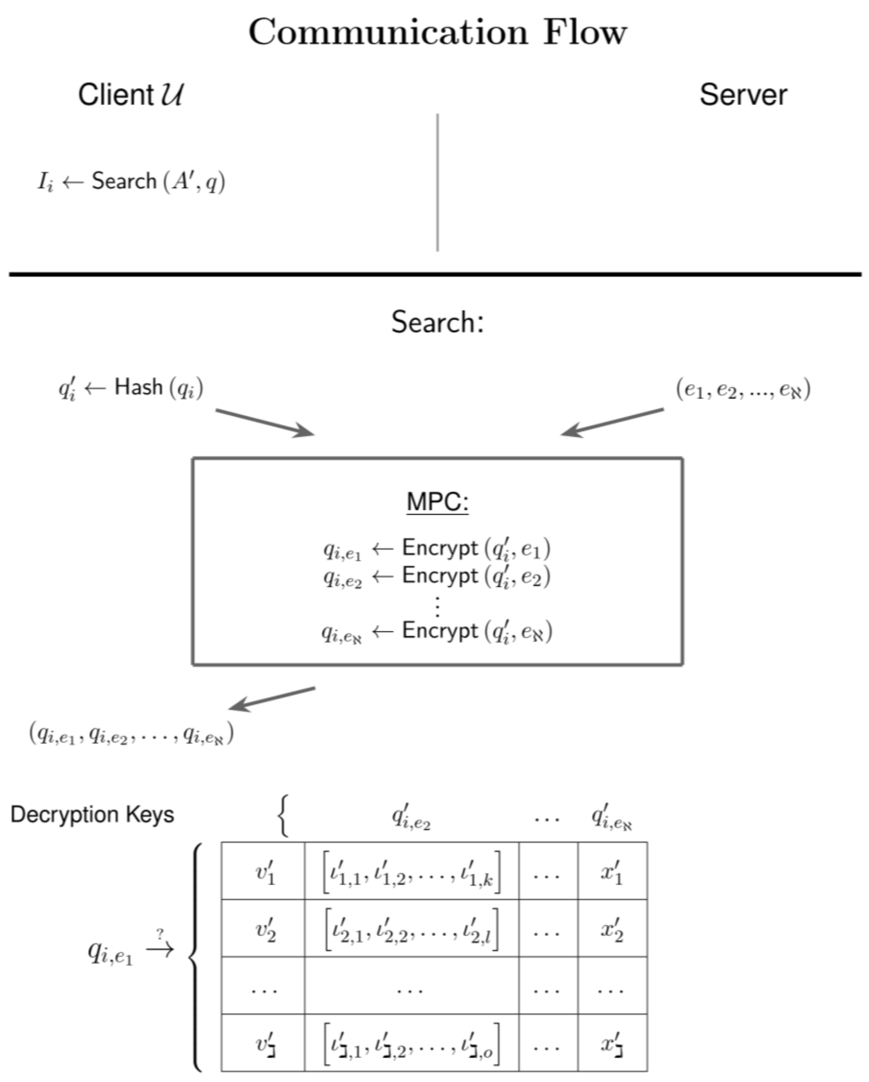
Garble:

Encryption Keys	e_1	Encrypt $(e_2, \text{Hash}(v_j))$	$\dots$	Encrypt $(e_{\aleph}, \text{Hash}(v_j))$
Inverted Index Matrix	Hash (v_1)	$[\iota_{1,1}, \iota_{1,2}, \dots, \iota_{1,k}]$	$\dots$	x_1
	Hash (v_2)	$[\iota_{2,1}, \iota_{2,2}, \dots, \iota_{2,l}]$	$\dots$	x_2
	$\dots$	$\dots$	$\dots$	$\dots$
	Hash $(v_{\beth})$	$[\iota_{\beth,1}, \iota_{\beth,2}, \dots, \iota_{\beth,o}]$	$\dots$	$x_{\beth}$

Keyword Search

Transfer

- We use generic MPC to encrypt the client's search query under the server's keys
- **Search** decrypt one row in the encrypted indexing to reveal the indices relevant to its search query:
    - The first variant of the encrypted search query is used to located the correct row in the encrypted indexing
    - The subsequent cells of the row are decrypted using the remaining encrypted variants of the search query.



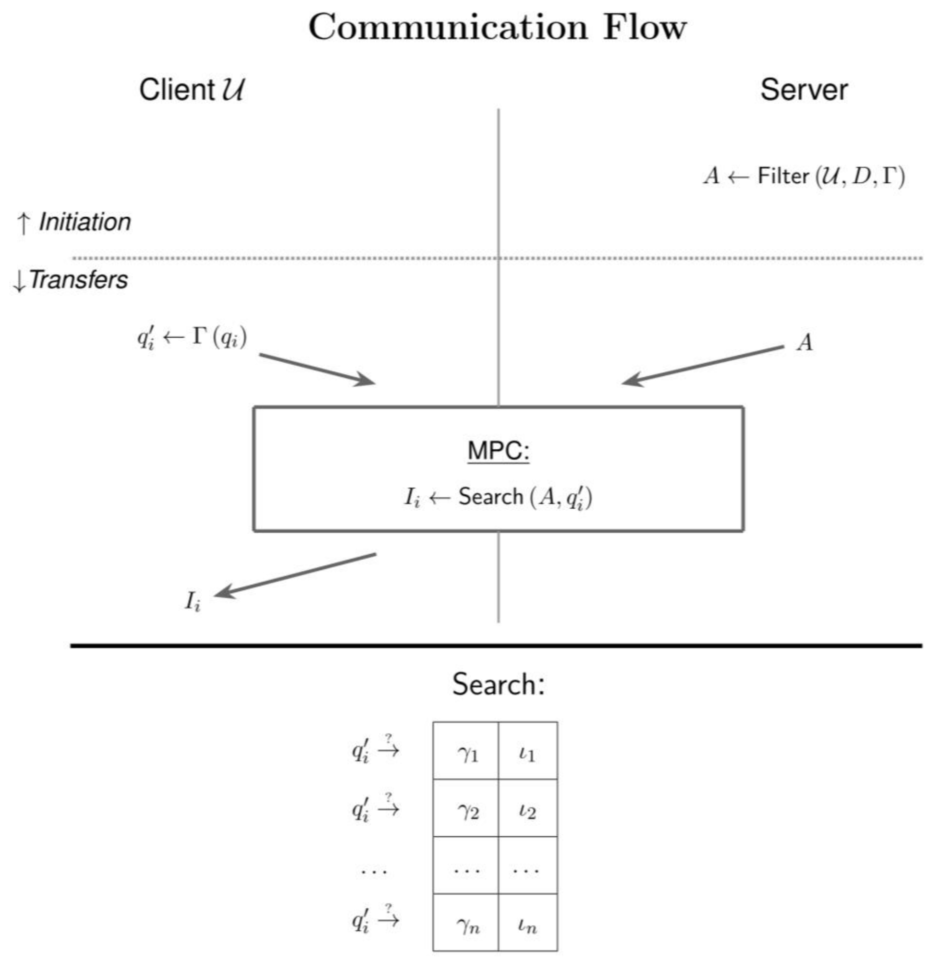
Communication Flow
Client $\mathcal{U}$
Server
$I_i \leftarrow \text{Search}\,(A', q)$
Search:
$q'_i \leftarrow \text{Hash}\,(q_i)$
$(e_1, e_2, ..., e_\aleph)$
MPC:
$q_{i,e_1} \leftarrow \text{Encrypt}\,(q'_i, e_1)$
$q_{i,e_2} \leftarrow \text{Encrypt}\,(q'_i, e_2)$
$\vdots$
$q_{i,e_\aleph} \leftarrow \text{Encrypt}\,(q'_i, e_\aleph)$
$(q_{i,e_1}, q_{i,e_2}, \dots, q_{i,e_\aleph})$
Decryption Keys
$q_{i,e_1} \overset{?}{\rightarrow}$

	q'_{i,e_2}	$\dots$	$q'_{i,e_\aleph}$
v'_1	$[\iota'_{1,1}, \iota'_{1,2}, \dots, \iota'_{1,k}]$	$\dots$	x'_1
v'_2	$[\iota'_{2,1}, \iota'_{2,2}, \dots, \iota'_{2,l}]$	$\dots$	x'_2
$\dots$	$\dots$	$\dots$	$\dots$
$v'_\beth$	$[\iota'_{\beth,1}, \iota'_{\beth,2}, \dots, \iota'_{\beth,o}]$	$\dots$	$x'_\beth$

Semantic Search

Initiation and Transfer

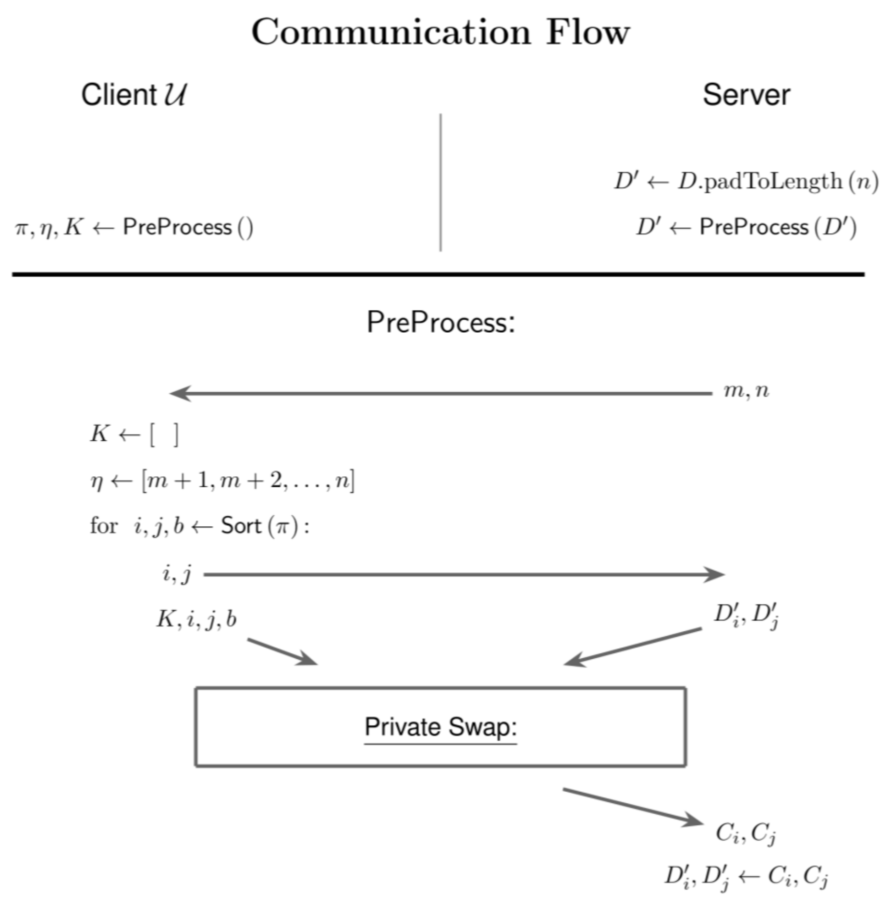
- We use a Large Language Model (LLM) to create vector embeddings
- **Filter** creates vector embeddings for each record and pairs them with an index.
- **Search** uses generic MPC to calculate the distances between the vector embedding of the search and the vector embeddings of the records, to find which indices are relevant



File Retrieval

Initiation

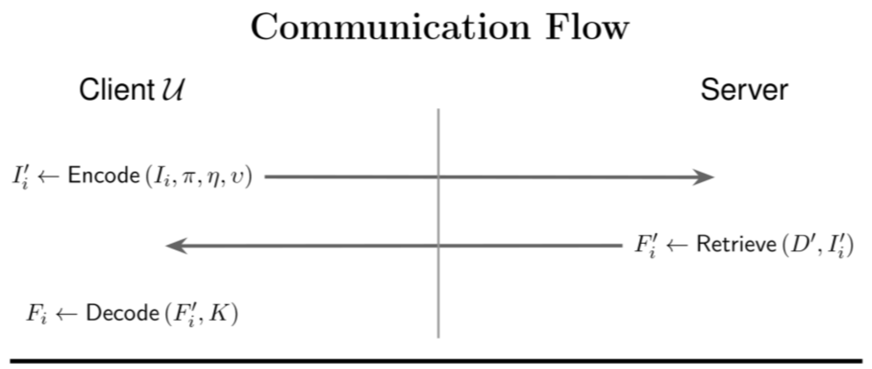
- The database is padded to length n with dummy items, where n is a multiple of 2
- **PreProcess** uses an oblivious sorting algorithm to sort a random permutation on the database and Private Swap (PS) to obliviously encrypt and swap records

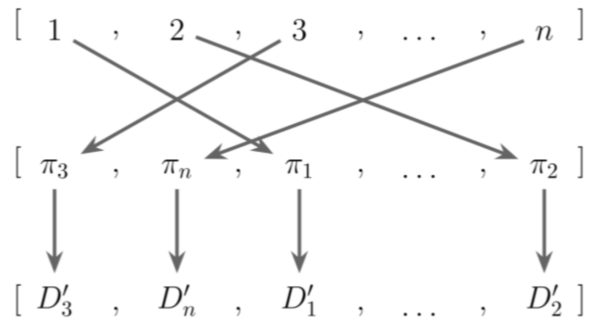

Communication Flow
Client $\mathcal{U}$
Server
$D' \leftarrow D.\text{padToLength}(n)$
$\pi, \eta, K \leftarrow \text{PreProcess}()$
$D' \leftarrow \text{PreProcess}(D')$
PreProcess:
m, n
$K \leftarrow [\ ]$
$\eta \leftarrow [m+1, m+2, \ldots, n]$
for $i, j, b \leftarrow \text{Sort}(\pi)$:
i, j
K, i, j, b
D'_i, D'_j
Private Swap:
C_i, C_j
$D'_i, D'_j \leftarrow C_i, C_j$

File Retrieval

Transfer

- **Encode** uses the random permutation to encode the indices and them to a fixed length using indices of dummy items
- **Retrieve** gets the encrypted record for the encoded indices
- **Decode** decrypts the retrieved records using the stored encryption keys


Communication Flow
Client $\mathcal{U}$
Server
$I_i' \leftarrow$ Encode $(I_i, \pi, \eta, \upsilon)$
$F_i' \leftarrow$ Retrieve (D', I_i')
$F_i \leftarrow$ Decode (F_i', K)


$[\ 1\ ,\ 2\ ,\ 3\ ,\ \ldots\ ,\ n\ ]$
$[\ \pi_3\ ,\ \pi_n\ ,\ \pi_1\ ,\ \ldots\ ,\ \pi_2\ ]$
$[\ D_3'\ ,\ D_n'\ ,\ D_1'\ ,\ \ldots\ ,\ D_2'\ ]$

Conclusion

Results

- Key findings:
  - Experimentally verify the support of a megabyte-sized database within a time budget of eight hours
  - **Encode** should be non-deterministic

- Future work:
  - Consider XSPIR for the retrieval of records
  - Implication of WLAN