Privacy Preserving Substring Search Protocol with Polylogarithmic Communication Cost

Nicholas Mainardi  Alessandro Barengghi Gerardo Pelosi

Email: name.surname@polimi.it

ACSAC 2019: San Juan (PR), December 11th, 2019
Data Confidentiality in Outsourcing Computation

Preserve Data Confidentiality in Outsourced Computation

Data Owner

Input Data

Cloud Service Provider

Results

Computation: ✓  Confidentiality against Cloud Service Provider: ×
Data Confidentiality in Outsourcing Computation

Preserve Data Confidentiality in Outsourced Computation

Data Owner

Cloud Service Provider

Input Data

Results

Computation: ?? Confidentiality against Cloud Service Provider: ✓

Problem: How to perform computation without disclosing data?

Nicholas Mainardi
Privacy-Preserving Substring Search: Overview

Entities
- Data owner with a document collection $\mathbf{D} = \{D_1, \ldots, D_z\}$ of $z \geq 1$ documents
- Untrusted server
- Users authorized to perform substring search queries over $\mathbf{D}$

Substring Search Queries
- Given a string $q$, find positions of all the repetitions of $q$ in each document of $\mathbf{D}$
- An inverted index for $\mathbf{D}$ is usually constructed for performance reasons

Leakage $\mathcal{L}$ of the protocol = information learnt by the untrusted server
Privacy-Preserving Substring Search: Phases

Setup

Data Owner

Index

Server

Encrypted D

Encrypted inverted index for D

Authorized Users

Nicholas Mainardi
Privacy-Preserving Substring Search: Phases

Query

Authorized User

Encrypted Query

CG

Server

Index

\[ D_1 = \text{ACGTCG} \]

\[ D_2 = \text{TCGAGT} \]

\[ D_3 = \text{CCAGT} \]

Encrypted Result

\[ S_1 = \{2, 5\} \]

\[ S_2 = \{2\} \]

\[ S_3 = \emptyset \]
Challenges in Multi-User Scenario

Enabling multiple simultaneous queries from distinct users:

- Without interaction between any user and the data owner
- Without synchronization among users
- Without a distinct copy of the outsourced index per user
- With **Collusion Resistance**: users colluding with the server cannot learn queries of non-colluding users

**Our Goal**

Design a **low bandwidth** privacy-preserving substring search protocol in multi-user scenario with information leakage $\mathcal{L}$ equal to:

- $n \approx \sum_{i=1}^{Z} |D_i|$: size of the document collection $D$
- $m = |q|$: length of the searched string $q$
- $occ$: overall number of occurrences of $q$ in $D$
Existing Solutions

Substring Searchable Symmetric Encryption Schemes

- Symmetric encryption schemes & Pseudo Random Functions employed to hide the content of both inverted index and queries
- Efficiency: ✓ → Bandwidth and computational cost of the queries independent from the size of the index
- Leakage \( \mathcal{L} \): × → Adversary observing queries learns search & access patterns

Search Pattern Leakage

I can distinguish if the same string is searched multiple times

Access Pattern Leakage

I can infer information on positions of occurrences of the searched string

Nicholas Mainardi
In all existing solutions, preventing search & access pattern leakage comes at a significant cost:

- Adoption of computationally heavy cryptographic primitives
- Bandwidth and computational costs dependent on the size of the outsourced index

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Underlying Crypto Primitive</th>
<th>Communication Cost</th>
<th>Multi-User Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wang et al. [WSLH17]</td>
<td>Pairings</td>
<td>$O(n)$</td>
<td>×</td>
</tr>
<tr>
<td>PBWT-Sec [SNR16]</td>
<td>Additive HE</td>
<td>$O((m+\text{occ})\sqrt{n})$</td>
<td>×</td>
</tr>
<tr>
<td>Ishimaki et al. [IHY17]</td>
<td>FHE</td>
<td>$O(C(m+\text{occ})\log(n))$</td>
<td>×</td>
</tr>
<tr>
<td>SA-ORAM [MB15]</td>
<td>ORAM</td>
<td>$\Omega(m\log^5(n)+\text{occ} \log^2(n))$</td>
<td>×</td>
</tr>
<tr>
<td>Our Protocol</td>
<td>PIR</td>
<td>$O((m+\text{occ})\log^2(n))$</td>
<td>✓</td>
</tr>
</tbody>
</table>

† the asymptotic cost hides a large constant factor $C$ due to usage of FHE, e.g., $C \geq 16 \times 10^6$, for providing 80-bit security parameters
Backward Search Algorithm (BSA)

For a string $s$ of length $n$ over an alphabet $\Sigma$:

- BSA locates occurrences of a string $q$ over $\Sigma$ in $s$ with $\mathcal{O}(m + \text{occ})$
- It employs a pre-computed inverted index made of 2 data structures:
  1. Matrix $M$ with $|\Sigma| \times n$ entries computed from the Burrows Wheeler Transform (BWT) of $s$
  2. Suffix Array (SA): store pointers to the $n$ suffixes of $s$, sorted in lexicographical order

From Single String to Document Collection

From $D = \{D_1, \ldots, D_z\}$, build a string $s = D_1\$D_2\$\ldots D_z\$:

Example: $D = \{\text{TCGAAGC}, \text{GAT}\}$, $q = \text{CGA}$, $s = T\text{CGAAGC}$\$\text{C$\times$GAT}$
Analyzing BSA in Our Scenario

In our scenario, $M$ and $SA$ are outsourced to a cloud server.

Data: $\text{Rank}$: dictionary of $|\Sigma|$ elements;
$\text{Rank}[c], c \in \Sigma \equiv \#$ of characters smaller than $c$ in $s$

1. \text{begin}
2. \hspace{1em} \text{c} \leftarrow q[m], \ \alpha \leftarrow \text{Rank}[c], \ \beta \leftarrow \alpha + M[c][n + 1]
3. \hspace{1em} \text{for } i \leftarrow m - 1 \text{ downto } 1 \text{ do}
4. \hspace{2em} \text{c} \leftarrow q[i], \ \text{r} \leftarrow \text{Rank}(c)
5. \hspace{2em} \alpha \leftarrow \text{r} + M[c][\alpha]
6. \hspace{2em} \beta \leftarrow \text{r} + M[c][\beta]
7. \hspace{1em} R_q \leftarrow \emptyset, \ \text{occ} \leftarrow \beta - \alpha
8. \hspace{1em} \text{for } i \leftarrow \alpha + 1 \text{ to } \beta \text{ do}
9. \hspace{2em} R_q \leftarrow R_q \cup \{SA[i]\}
10. \hspace{1em} \text{return } R_q

- Rank has $\mathcal{O}(|\Sigma|)$ entries $\rightarrow$ can be stored at client side
- Lightweight computation $\rightarrow$ can be performed at client side
- $2m + \text{occ}$ communication rounds with the server
Analyzing BSA in Our Scenario

In our scenario, $M$ and $SA$ are outsourced to a cloud server

Data: \textit{Rank}: dictionary of $|\Sigma|$ elements;

\begin{verbatim}
begin
\begin{align*}
  c &\leftarrow q[m], \quad \alpha \leftarrow \text{Rank}[c], \quad \beta \leftarrow \alpha + M[c][n+1] \\
  \text{for } i \leftarrow m - 1 \text{ downto } 1 \text{ do} \\
  &\quad c \leftarrow q[i], \quad r \leftarrow \text{Rank}(c) \\
  &\quad \alpha \leftarrow r + M[c][\alpha] \\
  &\quad \beta \leftarrow r + M[c][\beta] \\
  R_q &\leftarrow \emptyset, \quad \text{occ} \leftarrow \beta - \alpha \\
  \text{for } i \leftarrow \alpha + 1 \text{ to } \beta \text{ do} \\
  &\quad R_q \leftarrow R_q \cup \{SA[i]\}
\end{align*}
\end{verbatim}

\begin{itemize}
  \item \textit{Rank} has $O(|\Sigma|)$ entries $\rightarrow$ can be stored at client side
  \item Lightweight computation $\rightarrow$ can be performed at client side
  \item $2m + \text{occ}$ communication rounds with the server
    \begin{itemize}
      \item Problem: We cannot leak to the server which entries of $M$ and $SA$ are accessed by the client
    \end{itemize}
\end{itemize}
Our Solution: A PIR Protocol

Remote Access to Outsourced Data

Constraints:

- Low bandwidth and little computational overhead at client side
- The server cannot learn which entry is accessed by the client

Crypto primitive for our needs: **Private Information Retrieval**

![Diagram of PIR protocol]

Nicholas Mainardi
Which PIR for our Substring Search Protocol?

Lipmaa’s hierarchical PIR!

Motivations

1. Low bandwidth: \( O(\log^2(N)) \)
2. No costly Fully Homomorphic Encryption (FHE)
   - Hinge upon length-flexible additive homomorphic encryption (LFAHE) scheme \( \rightarrow \) Significant saving in computational effort

- We employ Damgård-Jurik LFAHE scheme
- Computational costs: \( O(N) \) server side, \( O(\log^5(N)) \) client side
**Problem for Simultaneous Queries**

$u \geq 1$ simultaneous queries requires $O(N + u \cdot N)$ memory at server side!

- Same memory consumption of having a distinct copy of the dataset per query
- Scalability issue

\[ \Downarrow \]

**Solution:** We propose different scheduling of operations to achieve memory consumption $O(N + u \cdot \log^2(N))$ without introducing any computational overhead.
Our Privacy-Preserving Substring Search Protocol: Setup

Data Owner:
- Arranges both $M$ and $SA$ as arrays for Lipmaa’s PIR and encrypts them entry-wise with a semantically secure block cipher
- Outsources data to untrusted server
- Shares Rank dictionary & block cipher key with authorized users

Performance
- Computing and encrypting $M$ and $SA$ costs $O(n)$ to data owner
- They have $O(n)$ entries $\rightarrow O(n)$ bandwidth
Our Privacy-Preserving Substring Search Protocol: Query

Each authorized user:

- Generates its own LFAHE Damgård-Jurik key-pair
- Executes backward search algorithm employing Lipmaa’s PIR to access $M$ and $SA$

**Performance**

Backward search $\equiv \mathcal{O}(m + occ)$ PIR queries:

- Bandwidth: $\mathcal{O}(\log^2(n)(m + occ))$
- User Cost: $\mathcal{O}(\log^5(n)(m + occ))$
- Server Cost: $\mathcal{O}(n(m + occ))$
## Leakage Analysis

- Size of $M$ and SA leaks $n$
- # of iterations of backward search leaks $m$ and $occ$
- No other information leakage thanks to PIR protocol (formally proven)

\[ L = n, m, occ \]

**Collusion Resistance:** ✓

- Each user employs its own Damgård-Jurik key-pair
- No more information than $L$ leaked to other users
Publicly available implementation in C/C++

**Experimental Setup:**
- Intel Xeon CPU E5-2620 with 16 cores & 32 threads, 128 GiB RAM
- 64-bit Gentoo Linux 17.0 OS

**Our Use Case:**
Paternity test based on positions of restriction enzyme sites in DNA:

$$q = CTGCAG$$

- DNA A and DNA B are blood relatives

- Our genomic dataset: the 21st human chromosome (40 MiB)
Experimental Results: Bandwidth

Bandwidth of a single PIR query:

Short bandwidth (few hundreds KiB) even for big dataset sizes

Nicholas Mainardi
Memory footprint for simultaneous queries:

Memory consumption growth rate marginally depends on dataset size

⇓

Significant memory savings w.r.t. having a distinct copy of index per query
Locate 1 occurrence of the substring $q = \text{CTGCAG}$:

- $Q_{\text{num}}$ phase: Find # of occurrences
- $Q_{\text{occ}}$ phase: Find positions of the occurrences

- 5 minutes to get # of occurrences in whole chromosome with 21 threads
- 5 additional minutes for each occurrence to retrieve its position
**Experimental Results: Server Computation**

Locate 1 occurrence of the substring $q = CTGCAG$:

- **Qnum phase**: Find \# of occurrences
- **Qocc phase**: Find positions of the occurrences

### Graph

- **Server Cost (min)**
- **Genome size (MiB)**

**Legend**

- Qnum 1 core
- Qnum multi-core
- Qocc 1 core
- Qocc multi-core

<table>
<thead>
<tr>
<th></th>
<th>Our Protocol</th>
<th>FHE Based Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
<td>Xeon E5-2620</td>
<td>Xeon E5-1620</td>
</tr>
<tr>
<td><strong>Threads</strong></td>
<td>21</td>
<td>72</td>
</tr>
<tr>
<td><strong>Genomic Dataset (MiB)</strong></td>
<td>$\approx 40$</td>
<td>$\approx 0.5$</td>
</tr>
<tr>
<td><strong>Substring Length</strong></td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td><strong>Query Exec. Time (min)</strong></td>
<td>5</td>
<td>30</td>
</tr>
</tbody>
</table>

Nicholas Mainardi
Concluding Remarks

Summary

- We propose a multi-user privacy-preserving substring search protocol with $O(\log^2(n)(m + occ))$ communication cost and no search & access patterns leakage.
- Our protocol combines backward search with PIR: improvements to PIR apply to our protocol too.
- Multi-core implementation achieves significant performance improvements on a real world use case.

Further Developments

Analysis of several recently proposed PIRs for our protocol:
- In this work, we aim at minimizing communication cost.
- Overall latency, including server computation, should be considered.
...and so, in conclusion, the proposed method...

THANK GOODNESS, ALMOST OVER... HOPEFULLY I DIDN'T BORE THEM TO TEARS.

...thank you, you've been a great audience...

OK, THE OBLIGATORY CALL FOR QUESTIONS AND I AM DONE...

any ques...

...tions?

OR NOT.

www.phdcomics.com

