Waldo: A Private Time-Series Database from Function Secret Sharing

Emma Dauterman, Mayank Rathee, Raluca Ada Popa, and Ion Stoica
UC Berkeley

IEEE Security & Privacy 2022
Time-series data

Growing trend: Rely on cloud databases to store and query time-series data

Challenge: Data is often sensitive

Danger of server compromise

Smart home  Industrial IoT  Remote patient monitoring
Time-series data: remote patient monitoring

\[
\begin{align*}
t & = 2022:03:00, \\
systolic & = 110, \\
diastolic & = 70, \\
heart \_ rate & = 80 \\
t & = 2022:03:01, \\
systolic & = 112, \\
diastolic & = 68, \\
heart \_ rate & = 72 \\
t & = 2022:03:02, \\
systolic & = 108, \\
diastolic & = 72, \\
heart \_ rate & = 76
\end{align*}
\]
Time-series data: remote patient monitoring

\[
\begin{align*}
time &= 2022:03:03, \\
systolic &= 106, \\
diastolic &= 75, \\
heart_rate &= 78
\end{align*}
\]

Patient → Database

\[
\begin{align*}
time &= 2022:03:03, \\
systolic &= 106, \\
diastolic &= 75, \\
heart_rate &= 78
\end{align*}
\]

Doctor

```
SELECT AVG(heart_rate) FROM MedicalHistory
WHERE (time BETWEEN 2022:03:01 and 2022:03:02)
AND (systolic > 110 OR diastolic > 80)
```

<table>
<thead>
<tr>
<th>time</th>
<th>systolic</th>
<th>diastolic</th>
<th>heart_rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>2022:03:00</td>
<td>110</td>
<td>70</td>
<td>80</td>
</tr>
<tr>
<td>2022:03:01</td>
<td>112</td>
<td>68</td>
<td>72</td>
</tr>
<tr>
<td>2022:03:02</td>
<td>108</td>
<td>72</td>
<td>76</td>
</tr>
</tbody>
</table>
Time-series data: remote patient monitoring

Patient

Doctor

SELECT AVG(heart_rate) FROM MedicalHistory
WHERE (time BETWEEN 2022:03:01 and 2022:03:02)
  AND (systolic > 110 OR diastolic > 80)

**Security: Protect data contents, query filter values**
Time-series data: remote patient monitoring

Patient

SELECT AVG(heart_rate) FROM MedicalHistory
WHERE (time BETWEEN '2022:03:01' and '2022:03:02')
AND (systolic > 110 OR diastolic > 80)

Doctor

Security: Protect data contents, query filter values

<table>
<thead>
<tr>
<th>time</th>
<th>systolic</th>
<th>diastolic</th>
<th>heart_rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>2022:03:00</td>
<td>110</td>
<td>70</td>
<td>80</td>
</tr>
<tr>
<td>2022:03:01</td>
<td>112</td>
<td>68</td>
<td>72</td>
</tr>
<tr>
<td>2022:03:02</td>
<td>108</td>
<td>72</td>
<td>76</td>
</tr>
</tbody>
</table>

Database

time = 2022:03:03,
systolic = 106,
diastolic = 75,
heart_rate = 78
Prior work on encrypted databases [CryptDB, Arx, BlindSeer, …]

Leakage: Server learns information that can be exploited to recover the data and/or query [IKK12, KKKO16, ZKP16, GLMP19, …].

Our goal: Protect search access patterns
- How server accesses data should not reveal info about data or query filter values
Finding a solution

Challenge: tools for hide search access patterns are expensive
- e.g. ORAM [GO96, PathORAM], MPC [GMW87, BGW88]
Threat model
Threat model

Waldo protects data contents, query filter values, and search access patterns.

Database schema, query structure

\[ \leq 1 \text{ (malicious) compromise} \]

Clients

Servers
Waldo protects data contents, query filter values, and search access patterns

Patient

Doctor

\[
\begin{align*}
\text{time} & = 2022:03:03, \\
\text{systolic} & = 106, \\
\text{diastolic} & = 75, \\
\text{heart_rate} & = 78
\end{align*}
\]

SELECT AVG(heart_rate) FROM MedicalHistory
WHERE (time BETWEEN 2022:03:01 and 2022:03:02)
    AND (systolic > 110 OR diastolic > 80)

<table>
<thead>
<tr>
<th>time</th>
<th>systolic</th>
<th>diastolic</th>
<th>heart_rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>2022:03:00</td>
<td>110</td>
<td>70</td>
<td>80</td>
</tr>
<tr>
<td>2022:03:01</td>
<td>112</td>
<td>68</td>
<td>72</td>
</tr>
<tr>
<td>2022:03:02</td>
<td>108</td>
<td>72</td>
<td>76</td>
</tr>
</tbody>
</table>
Waldo protects data contents, query filter values, and search access patterns

SELECT AVG(F4) FROM MedicalHistory
WHERE (F1 BETWEEN 2022:03:01 and 2022:03:02) and (F2 > 110 OR F3 > 80)

Protects search access patterns (can be exploited to recover data/query).
Notation

Split \( x \in 2^\ell \) into secret shares \([x]_1, \ldots, [x]_n\) such that \( \sum_{i=1}^{n} [x]_i = x \)
Tool: Function Secret Sharing (FSS)  [BGI15, BGI16, BGI19, BCG+21]

Two-party FSS scheme for function $f$ splits function into 2 succinct shares (“keys”). Given a key, parties can compute shares of $f(x)$ without learning $f$.

Syntax:
- $\text{Gen}(f) \rightarrow K_1, K_2$: Generate keys for function $f$ that hide $f$
- $\text{Eval}(K, x) \rightarrow y$: Given key and input value $x$, output $y = [f(x)]$

Efficient constructions exist for point functions and comparison functions.
Aggregate queries for public data

SELECT SUM(cost) FROM MedHistory
WHERE (ID = 112)

$K_1, K_2 \leftarrow \text{Gen}(f_{112\rightarrow 1})$

Output 1700

Challenge: How do we extend these techniques to private data?
Idea: leverage search index structure

Entry \((i, j) = 1\) if record \(i\) has heart_rate \(j\), 0 otherwise

<table>
<thead>
<tr>
<th>Record #</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Record 3 has heart_rate = 79
Idea: leverage search index structure

<table>
<thead>
<tr>
<th>Record #</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>heart_rate</td>
<td>79</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Appends are cheap $\rightarrow$ append to table

Target apps querying recent data

Limitation: Constrains domain of filtered features
- In applications we examined, features in predicates already from small domain (e.g. $2^8$) or can easily be mapped to one
  - EX: age, height, heart rate, blood glucose level, percentage
- Represent large domains via conjunctions
- Only restricts filtered features (not aggregated features).
Idea: leverage search index structure

SELECT COUNT(*) FROM MedHistory
WHERE (heart_rate = 80)

\(K_1, K_2 \leftarrow \text{Gen}(f_{80 \rightarrow 1})\)

Output 2
Replicated secret sharing (RSS) with FSS

\[ \begin{align*}
K_1^{(1)}, K_2^{(1)} & \leftarrow \text{Gen}(f) \\
K_1^{(2)}, K_2^{(2)} & \leftarrow \text{Gen}(f) \\
K_1^{(3)}, K_2^{(3)} & \leftarrow \text{Gen}(f) \\
\text{Output} & = \sum_{i=1}^{6} r_i
\end{align*} \]

[BKK+20]
Other contributions (see paper)

- Support for filtering based on multiple predicates
- Tools for cheap malicious security
- Construction for complex aggregates over time ranges
  + Supports non-linear aggregation functions (e.g. min, max, top-k)
  - Only filters on time
Evaluation

Code available at: https://github.com/ucbrise/waldo

Experiment setup:
• Three 32-core servers
• 3 Gbps connection with 20ms RTT
• 8-predicate queries

Baselines (see paper):
• Oblivious multidimensional tree (R-tree with PathORAM)
• Generic MPC: MP-SPDZ (3PC honest-majority)
Latency

Point and range almost identical

Grows $\text{polylog}(N)$

Grow linearly with $N$
Throughput

![Graph showing throughput (ops/s) vs. Records (N)]

- Waldo (point)
- Waldo (range)
- ORAM

Throughput (ops/s) vs. Records (N):
- 303x improvement for Waldo (point)
- 22x improvement for Waldo (range)

23
Conclusion

Waldo is a time-series database that provides:

- **Functionality**: Supports multi-predicate queries
- **Security**: Hides data contents, query filter values, and search access patterns
- **Efficiency**: Low latency, high throughput, low bandwidth

Thanks!

Emma Dauterman
edauterman@berkeley.edu

https://github.com/ucbrise/waldo