Revisiting Square Root ORAM
Efficient Random Access in Multi-Party Computation

Samee Zahur
Jack Doerner
David Evans

Xiao Wang
Jonathan Katz

Mariana Raykova

Adrià Gascón

oblivc.org/sqoram
Secure multi-party computation applications

- Set intersection [FNP04]
- Iris code matching [LCPLB12]
- Median computation [AMP04]
- Linear ridge-regression [NWIJBT13]

Matrix factorization for recommendations [NIWJTB13]
Random Access

void oscrypt_smix(obliv uint8_t * B, s ...  
for (i = 0; i < N; i += 2) {  
j = integerify(X, r) & (N - 1);
    temp = V2[j];
    xorBits(X, temp, 32*r);
    oscrypt_blockmix_salsa8(X, Y, r);

    j = integerify(Y, r) & (N - 1);
    temp = V2[j];

    for (size_t jj = 0; jj < 32 * r; j  
        Y[jj] ^= temp[jj];
    }
}
Hiding access pattern

**Linear scan**

- Access every element
- Per-access cost: $\Theta(n)$

**Oblivious RAM**

- Continually shuffle elements around
- Per-access cost: $\Theta(\log^p n)$
Figure from: Wang, Chan, Shi. **Circuit Oram.** CCS’15
Approach: revisit old schemes


Considered slow for MPC because of per-access hash evaluation.

Per-access amortized cost: $\Theta(\sqrt{n \log n})$
Four-element ORAM

Larger Sizes

\[ \Pi_1 \]
\[
\begin{array}{cccccccc}
4 & 1 & 5 & 2 & 0 & 7 & 3 & 6 \\
\end{array}
\]

\[ \Pi_A \rightarrow \]
\[
\text{Permute}
\]

\[ \Pi_A \cdot \Pi_1 \]
\[
\begin{array}{cccccccc}
7 & 4 & 3 & 1 & 5 & 0 & 6 & 2 \\
\end{array}
\]
4-Block ORAM

Cost: $5B + B + 2B + 3B + \ldots$

$= 11B$ every 3 accesses
Comparison

**Linear scan**

Cost: $4B = \frac{12B}{3}$

**Our scheme**

Cost: $\frac{11B}{3}$
Four-element ORAM

Larger Sizes

$$\pi^1 \rightarrow \begin{array}{cccccccc}
4 & 1 & 5 & 2 & 0 & 7 & 3 & 6 \\
\end{array}$$

$$\pi_A \rightarrow \text{Permute}$$

$$\pi_A \cdot \pi^1 \rightarrow \begin{array}{cccccccc}
7 & 4 & 3 & 1 & 5 & 0 & 6 & 2 \\
\end{array}$$
Position map
Creating position map

\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
1 & 3 & 0 & 6 & 2 & 7 & 5 & 4 \\
\end{array}
\]

\begin{tabular}{cc}
\hline
x & \pi(x) \\
0 & 2 \\
1 & 0 \\
2 & 4 \\
3 & 1 \\
4 & 7 \\
5 & 6 \\
6 & 3 \\
7 & 5 \\
\hline
\end{tabular}
Creating position map

<table>
<thead>
<tr>
<th>x</th>
<th>π(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>
Inverse permutation

\[ \pi_C \cdot p \]

\[ \pi_C \cdot p = \pi_B \]

\( \pi_A \)

\( p \)

\( \text{Permuted} \)
Inverse permutation

Bob computes

\[ \pi_B = \pi_A \cdot p \]

\[ \pi_B^{-1} = p^{-1} \cdot \pi^{-1}_A \]
Rinse and repeat

1. Shuffle elements
2. Recreate position map
3. Service $T = \sqrt{n \log n}$ accesses
Access time

![Graph showing the running time per access compared to the number of blocks for different ORAM algorithms. The graph plots the running time on a logarithmic scale against the number of blocks, which increases exponentially. The x-axis represents the number of blocks, ranging from $2^3$ to $2^{19}$, while the y-axis represents the running time per access, ranging from $10^{-4}$ to $10^2$ seconds. Three algorithms are compared: Linear Scan, Circuit ORAM, and Square-Root ORAM. The Linear Scan algorithm has the highest running time per access, followed by Circuit ORAM, and then Square-Root ORAM, which shows the lowest running time per access with the smallest slope.]
Initialization cost

![Graph showing running time per init (seconds) vs. number of blocks for different ORAM methods: Circuit ORAM, Square-Root ORAM, and Linear Scan.]
## Benchmarks

<table>
<thead>
<tr>
<th>Task</th>
<th>Parameters</th>
<th>Linear scan</th>
<th>Circuit ORAM</th>
<th>Square-root ORAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary search</td>
<td>$2^{10}$ searches, $2^{15}$ elements</td>
<td>1020 s</td>
<td>5041 s</td>
<td>825 s</td>
</tr>
<tr>
<td>Breadth-first search</td>
<td>$2^{10}$ vertices, $2^{13}$ edges</td>
<td>4570 s</td>
<td>3750 s</td>
<td>680 s</td>
</tr>
<tr>
<td>Stable matching</td>
<td>$2^9$ pairs</td>
<td>-</td>
<td>189000 s</td>
<td>119000 s</td>
</tr>
<tr>
<td>scrypt hashing</td>
<td>$N = 2^{14}$</td>
<td>$\approx 7$ days</td>
<td>2850 s</td>
<td>1920 s</td>
</tr>
</tbody>
</table>
Conclusion

We revisited a well-known scheme and used it to

• Lower initialization cost
• Improve breakeven point

Shows that asymptotic costs are not the final word, concrete costs require more consideration.
Download

oblivc.org/sqoram

Contact for help:
Samee Zahur <samee@virginia.edu>