Oblivious Computation
Part II - Oblivious Sorts

Gilad Asharov
Bar-Ilan University

The 12th Bar-Ilan Winter School on Cryptography
Advances in Secure Computation
Access Patterns Reveal Information!
Models of Computation

Circuits

Emulate easily

$T^3 \log T$

[CR73,PF79]

$(T \gg N)$

RAM Model

Random Access Machine

CPU Operation

Memory access

Metrics:

Size (how many wires, gates)
Depth (parallelism)

Time
Size of the memory

$T$
$N$
Oblivious RAM Compiler: State of the Art

Lower bound: $\Omega(\log N)$

[GoldreichOstrovsky’96, LarsenNielsen’18]

Hierarchical

$O(\log N)$

Computational security

[OptORAMa’20]

Tree based ORAM

$O(\log^2 N)$

Statistical security

[PathORAM,CircuitORAM]
Example: Oblivious Sorts

- **Merge sort:** $O(n \log n)$
  - non oblivious

- **Bubble sort:** $O(n^2)$
  - oblivious

- **Other oblivious sorts?**
# Oblivious Sorts

In the RAM Model

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Oblivous</th>
<th>Client Storage</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Merge sort</td>
<td>No</td>
<td>$O(1)$</td>
<td>$2n \log n$</td>
</tr>
<tr>
<td>Bitonic sort</td>
<td>Yes</td>
<td>$O(1)$</td>
<td>$n \log^2 n$</td>
</tr>
<tr>
<td>AKS sort</td>
<td>Yes</td>
<td>$O(1)$</td>
<td>$5.4 \times 10^7 \times n \log n$</td>
</tr>
<tr>
<td>Zig-zag sort</td>
<td>Yes</td>
<td>$O(1)$</td>
<td>$8 \times 10^4 \times n \log n$</td>
</tr>
</tbody>
</table>

*constants of AKS, Zig-zag are from [Goodrich’14]

Z: poly log k

Most practical

For $n > 2^{30}$

x5 faster than Bitonic
Bucket Oblivious Sort

Oblivious Permute + Non-Oblivious Sort = Oblivious Sort

Merge-Sort
Bucket Oblivious Permute

- Interpret the input array as $B$ buckets of size $Z$ each ($Z=\text{poly} \log k$, $B=N/Z$, $k$ is the security parameter)
- Assign to each element a random destination bin $[1,\ldots,B]$
- Add dummy bins

(We later remove these dummy elements using the non-oblivious sort)
Bucket Oblivious Permute

MergeSplit - takes all read elements in input buckets and distribute them to output buckets according to the $i$th MSB

Overflows?
Each bucket (in expectation) is “half full”
# Oblivious Sorts

## In the RAM Model

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Oblivious</th>
<th>Client Storage</th>
<th>Runtime</th>
<th>Error Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Merge sort [vonNeumann’45]</td>
<td>No</td>
<td>$O(1)$</td>
<td>$2n \log n$</td>
<td>0</td>
</tr>
<tr>
<td>Bitonic sort [Batcher’68]</td>
<td>Yes</td>
<td>$O(1)$</td>
<td>$n \log^2 n$</td>
<td>0</td>
</tr>
<tr>
<td>AKS sort [AKS’83]</td>
<td>Yes</td>
<td>$O(1)$</td>
<td>$5.4 \times 10^7 \times n \log n$</td>
<td>0</td>
</tr>
<tr>
<td>Zig-zag sort [Goodrich’14]</td>
<td>Yes</td>
<td>$O(1)$</td>
<td>$8 \times 10^4 \times n \log n$</td>
<td>0</td>
</tr>
<tr>
<td>Bucket oblivious sort [ACNPRS’20]</td>
<td>Yes</td>
<td>$O(1)$</td>
<td>$2n \log n$</td>
<td>$\approx e^{-Z/6}$</td>
</tr>
<tr>
<td>Bucket oblivious sort [ACNPRS’20]</td>
<td>Yes</td>
<td>$O(1)$</td>
<td>$\approx 2n \log n \log^2 Z$</td>
<td>$\approx e^{-Z/6}$</td>
</tr>
</tbody>
</table>

*constants of AKS, Zig-zag are from [Goodrich’14]
(Oblivious) Sorting Faster Than $O(n \log n)$?
(Oblivious) **Sorting Faster than** $O(n \log n)$?

"**No!** Such a result is not possible with **comparison-based**"

Knuth73: The Art of Computer Programming

**Non-comparison based sorts?** $(k - \text{length of the key})$

Radix-sort $O(k \cdot n)$, counting sort $O(2^k + n)$

**RAM Model:**

Radix sort, counting sort - make input-dependent memory accesses

Do not have equally efficient counterparts in the circuit model
What about the Circuit Model?

Can we go lower than $(k + w) \cdot \Omega(n \log n)$ circuit-size?

**Comparator based?**
- Any comparator-based sorting circuit must consume $\Omega(n \log n)$ comparators
- Even for $k = 1$ long key!

**Stability?**
- Stable sort requires $\Omega(n \log n)$ selector gates in the *indivisible* model, even for $k = 1$ [Lin, Shi, Xie19]

Assuming a well-known network conjecture, sorting circuits of size $(k + w) \cdot o(n \log n)$ do not exist for general $k$ [Afshani, Freksen, Kamma, Larsen19]

No unconditional lower bound is known
\( n \): number of elements
\( k \): length of each key (#bits)
\( w \): payload (#bits)

\[ (k + w) \cdot O(n \log n) \]

*ignoring polylog* terms

\[ (k + w) \cdot O(nk) \]

Better for \( k \in o(\log n) \)

*Not Comparison Based!*

*Not Stable!*

Indeed worse than \( n \log n \) for general \( k \)
Tight Compaction
A Central Problem!

Tight Compaction

Circuit Model
- Linear size circuit (ignoring polylog* factors)
- (Almost) Linear size sorting circuit (short keys) [ALS’21]

RAM Model
- Linear time oblivious compaction
- Optimal Oblivious RAM compiler [OptORAMa, AKLNPS’20]
Tight Compaction

- **Input**: An array of size n where each element is marked 0 or 1
- **Output**: all 0-elements appear before 1-elements

- **RAM model? Trivial in O(n)**
  - Oblivious RAM model?
    - Deterministic: $O(n \log n)$ [AKS'83]
    - Open question from [LeightonMaSuel'95]
      - Reveals the number of 0's, randomized
    - Randomized: $O(n \log \log n)$
      - [MZ'14, LST'18] (negl error prob.)
    - **lower bound**: stable, indivisible, $\Omega(n \log n)$
      - [LST'18]

- **OptORAMa [AsharovKomargodskiLinNayakPessicoShi20]**:
  - Deterministic, $O(n)$, very large constant
  - Better constant [DittmerOstrovksy20]
  - $O(n)$ work, depth $O(\log n)$
    - [AsharovKomargodskiLinPessicoShi20]
Linear size circuit (ignoring polylog* factors)

- \( O(nw) \cdot \text{poly log}^* n \) size circuit
- Compacting \( n \) balls of size \( w \) each

Not comparison based

Not stable

Balls and bins model

Metadata is computed using bit-slicing tricks
1) Simple, randomized oblivious tight compaction (RAM model) in $O(n \log \log k)$
   Lin, Shi, Xie: Can we Overcome the $n \log n$ Barrier for Oblivious Sorting? SODA’19

2) Linear size compaction circuit
   Asharov, Lin, Shi: Sorting Short Keys in Circuits of Size $o(n \log n)$, SODA’21

3) Linear time algorithm for oblivious tight compaction in the RAM model
   Asharov, Komargodski, Lin, Nayak, Pesci and Shi: OptORAMa, Optimal Oblivious RAM, EUROCRYPT’20
Simple Oblivious Tight Compaction
[Lin, Shi, Xie, SODA’19]

0 1 1 0 0 1 0 1 0 0 1 1 0 0 0 1 0 0 1 0 1 1 0 0 1 0 1 0 1 1 0 1 0 1 1 1

1) Interpret the array as n/Z bins of size Z each
Simple Oblivious Tight Compaction
[Lin, Shi, Xie, SODA’19]

1) Interpret the array as n/Z bins of size Z each

\[
\begin{array}{cccccc}
0 & 1 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 & 1 & 1
\end{array}
\]

2) Random shift cycle for each row

\[Z \in \text{polylog}(k) = \log^6 k\]
Simple Oblivious Tight Compaction
[Lin, Shi, Xie, SODA’19]

1) Interpret the array as \( \frac{n}{Z} \) bins of size \( Z \) each

\[
\begin{array}{cccccccc}
0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\
\end{array}
\]

\[
Z \in \text{polylog}(k) = \log^6 k
\]

2) For each row, perform random shift cycle

\[
\begin{array}{cccccccc}
0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
\end{array}
\]

\( O(n) \)
Simple Oblivious Tight Compaction
[Lin, Shi, Xie, SODA’19]

1) Interpret the array as $n/Z$ bins of size $Z$ each

2) For each row, perform random shift cycle

3) Sort each column independently

$$Z = \log^6 k$$

$$\frac{n}{Z} \cdot Z \log^2 Z = O(n \log^2 Z) \in O(n \log \log k)$$
Simple Oblivious Tight Compaction
[Lin, Shi, Xie, SODA’19]

Claim:
W.h.p, there exists a mixed stripe of size $Z/\log^2 k = \log^4 k$ rows

1) Interpret the array as $n/Z$ bins of size $Z$ each
2) For each row, perform random shift cycle
3) Sort each column independently
4) copy the mixed stripe to some working array of size $\frac{n}{\log^6 k} \cdot \log^4 k = n/\log^2 k$
5) Sort (obliviously!) the mixed stripe; write it back
1) Interpret the array as $n/Z$ bins of size $Z$ each
2) For each row, perform random shift cycle
3) Sort each column independently
4) Copy the mixed stripe to some working array of size $\frac{n}{\log^6 k} \cdot \log^4 k = \frac{n}{\log^2 k}$
5) Sort (obliviously!) the mixed stripe; write it back

When $n \in O(k)$, the algorithm is statistically secure and runs in $O(n \log \log n)$
1) Simple, randomized oblivious tight compaction (RAM model) in $O(n \log \log k)$
   Lin, Shi, Xie: Can we Overcome the $n \log n$ Barrier for Oblivious Sorting? SODA’19

2) Linear size compaction circuit
   Asharov, Lin, Shi: Sorting Short Keys in Circuits of Size $o(n \log n)$, SODA’21

3) Linear time algorithm for oblivious tight compaction in the RAM model
   Asharov, Komargodski, Lin, Nayak, Peserico and Shi: OptORAMa, Optimal Oblivious RAM, EUROCRYPT’20
Our Cost Model

Generalized Boolean gates

$w$-selector gates

Selector

Reverse Selector

"Reverse" selector
Oblivious Tight Compaction

- **Input**: An array of size n where each element is marked 0 or 1
- **Output**: all 0-elements appear before 1-elements

(1) Count the number of balls marked 0

(2) Mark the elements that are “misplaced”

Observation: number of reds always equals number of blues
We just have to swap them!
Loose Swap

Bipartite Expander Graph
- $O(1)$ degree
- Constant spectral expansion
Loose Swap

1 that wants to swap with 0
0 that wants to swap with 1

For every pair of

\[ O(nd^2) \text{ boolean gates} \]
\[ O(n \cdot d^2) \text{ w-selector gates} \]

\[ O(nw) \text{ gates total} \]
Claim

At the end of this procedure, there are no more than $n/100$ remaining swaps.

- Consider the sets of “survivors”
- Each of size $> n/200$
  (Recall $#\text{reds}=#\text{blues}$)
- Their sets of neighbors must be disjoint

- Expansion property:
  for any set of size $> n/200$, number of neighbors $> n/2$
Loose Swap

Loose Compactor

Loose Compaction

Tight Compaction
Loose Swap

Loose Compactor

Recurse
Loose Swap

Loose Compactor

Recurse
Loose Swap

Reverse Route

Recurse

L

R

1%
What Do We Have So Far?

Loose Swap + Loose Compactor $\implies$ Tight Compactor

$O(nw)$ $\quad O(n \cdot f(n) + nw)$ $\quad O(n \cdot f(n) + nw)$
Loose Compactor

Bipartite Expander Graph

- $O(1)$ degree
- Constant spectral expansion

Two stages:

- Find which edges will be chosen
- Find a matching (routes)
- Route elements on those edges

$O(n \log n)$ boolean gates

$O(n)$ $w$-selector gates

$O(n \log n + nw)$ boolean gates

$\leq 1\%$ are marked

Non comparison based!
Find a Matching

Repeat $\log n$ times:

- All blue red in L: propose to all incident vertices
- R: accept only if 1 proposal received, else reject all
- All blue red in L: if received “accept” become

$O(n \log n)$ boolean gates

Each edge has an indicator (a wire) whether to “route” an element on it

Offline route-finding does not depend on the payload!
Route

On all marked edges:
• Move an element
  Requires $O(n)$ $w$-selector gates

$O(n \log n)$ Boolean gates

Overall circuit requires $O(n \log n + nw)$ Boolean gates
What Do We Have So Far?

Loose Compactor $O(n \log n + nw)$

Loose Swap + Loose Compactor $O(nw)$ $\implies$ Tight Compactor $O(n \cdot f(n) + nw)$

Tight Compactor $O(n \log n + nw)$

Key insight: don’t stop here
Tight Compactor $O(n \log n + nw) \implies$ Loose Compactor $O(n \log \log n + nw)$

- **Dense**
- **Sparse**

**Chunk:** $\log n$ balls \hspace{1cm} (each ball $w$ bits long)

$n/\log n$ chunks, $w \log n$ bit payload each

**Step 1:** Run tight compact to move all “dense” chunks to the front \hspace{1cm} $O(nw)$ circuit

Very few chunks are dense
Tight Compactor $O(n \log n + nw) \implies$ Loose Compactor $O(n \log \log n + nw)$

Dense  Sparse

Step 1: Run tight compact to move all “dense” chunks to the front

Step 2: Run tight compact on all sparse chunks

Very few chunks are dense

$n/\log n \times \log n$ balls of size $w$-bit each

Tight compact: $\log n \cdot \log \log n + \log n \cdot w$

$O(n \log \log n + nw)$ size circuit
Tight Compactor $O(n \cdot f(n) + nw)$ $\implies$ Loose Compactor $O(n \cdot f(f(n)) + nw)$

**Dense**

**Sparse**

Very few chunks are dense

**Chunk:** $f(n)$ balls $(w \cdot f(n)$ bits)

$n/f(n)$ chunks, $w \cdot f(n)$ bit payload each

**Step 1:** Run tight compact to move all "dense" chunks to the front $O(nw)$ circuit

```
  ●●●●●      ●●●●●      ●●●●●      ●●●●●      ●●●●●
  ●●●●●      ●●●●●      ●●●●●      ●●●●●      ●●●●●
```

BIU

Center for Research in Applied Cryptography and Cyber Security
Dense \quad \rightarrow \quad Sparse

Step 1: Run tight compact to move all “dense” chunks to the front

Step 2: Run tight compact on all sparse chunks

Very few chunks are dense

Tight Compactor $O(n \cdot f(n) + nw) \quad \implies \quad$ Loose Compactor $O(n \cdot f(f(n)) + nw)$

$n/f(n) \times f(n)$ balls of size $w$-bit each

Tight compact: $f(n) \cdot f(f(n)) + f(n) \cdot w$

$O(n \cdot f(f(n)) + nw)$ size circuit
What Do We Have So Far?

Loose Compactor $O(n \log n + nw)$

Loose Compactor $O(n \cdot f(n) + nw)$ $\implies$ Tight Compactor $O(n \cdot f(n) + nw)$

Tight Compactor $O(n \cdot f(n) + nw)$ $\implies$ Loose Compactor $O(n \cdot f(f(n)) + nw)$

Tight Compactor $O(n \cdot f(n) + nw)$ $\implies$ Tight Compactor $O(n \cdot f(f(n)) + nw)$
Bootstrapping!

Tight Compactor $O(n \log n + nw)$

A tight compactor of size: $\text{poly}(\log^* n - \log^* w) \cdot O(nw)$
Tight Compaction

Selection

Sorting
Tight Compaction
A Central Problem!

- Linear size circuit (ignoring polylog* factors)
- (Almost) Linear size sorting circuit (short keys) [ALS'21]
- Linear time compaction
- Optimal Oblivious RAM compiler
  [OptORAMa, AKLNPS'20]
Oblivious RAM Compiler: State of the Art

Lower bound: $\Omega(\log N)$
- [GoldreichOstrovsky‘96, LarsenNeilsen’18]

Hierarchical
- $O(\log N)$
  - [O90, GO96]
  - Computational security
    - [OptORAMa’20]

Tree based ORAM
- $O(\log^2 N)$
  - Statistical security
    - [Shi, Chan, Stefanov11]
Thank You!