Oblivious Algorithms

Brice Minaud

e-mail: brice.minaud@inria.fr

MPRI, 2021
Course outline

“Techniques in Cryptography and Cryptanalysis”: will cover (a choice of) important areas of cryptography.

- Lattices ✔️ done
- Zero-knowledge proofs ✔️ done
- Oblivious algorithms now
Outsourcing storage

With SNARKs: could outsource computation.

Now: outsource storage.

The host server is honest-but-curious.
Outsourcing storage

**Setting:**

Client stores $n$ data items.

“Key-value store”: each item is pair (address, data).

Client makes read/write operations:

- **read($a$)**: read data at address $a$; or
- **write($a,d$)**: write data $d$ at address $a$.

Server sees accessed addresses, and data read/written.
Encryption

First, want to encrypt all data.

Before writing any data, encrypt it with semantically secure (IND-CCA) symmetric encryption.

→ The server learns nothing about the content of any data.
Including whether two data items are equal!
Symmetric Encryption.

Message space $\mathbf{M}$, ciphertext space $\mathbf{C}$, key space $\mathbf{K}$.

**Setup:** Pick key $K \leftarrow \$ K.

**Encryption:** encryption of $M \in \mathbf{M}$ is $C = \text{Enc}_K(M) \in \mathbf{C}$.

**Decryption:** decryption of $C$ is $M = \text{Dec}_K(C)$.

**Correctness:** for all $M \in \mathbf{M}$,

$$\text{Dec}_K(\text{Enc}_K(M)) = M.$$

**Security:** here, let us say IND-CCA.

**Caveats:**

- Deterministic scheme cannot be IND-CCA. Need randomness, or nonces.

- “Security” above only covers confidentiality, not integrity.
IND-CCA: indist. under Chosen-Ciphertext Attacks

Adversary

- Pick $M'_i$.
- Pick $C_i$.
- Pick $M_1, M_2$.
- Pick $M''_i, C'_i$... with $C'_i \neq C^*$
- Compute $b'$.

Challenger

- Pick $K \leftarrow \{0,1\}^k$.
- $M'_i$ → $\gets \text{Enc}_K(M'_i)$
- $C_i$ → $\gets \text{Dec}_K(C_i)$
- $M_1, M_2$ → $\gets \text{Enc}_K(M_b)$
- $C^* = \text{Enc}_K(M_b)$
- Pick $b \leftarrow \$ \{0,1\}$
- Repeat freely
**Conclusion:** hiding the content of data is easy with encryption.

But the server also sees the address accessed by the client: “access pattern”.

This reveals a lot of information about the client’s activities.
Simple example

Client = hospital, storing patient files.

Client often searches for all patients with age between $x$ and $y$.

What the server sees:

1. Client accesses files $a$, $b$, $c$.
2. Later, client accesses files $b$, $c$, $d$.

This is the only possible configuration (up to symmetry)!

→ Server learns that age of files $b$, $c$ is between $a$ and $d$. 
Simple example

1. Client accesses files a, b, c.
2. Client accesses files b, c, d.
3. Client accesses files c, d.

Then the only possible order of ages is a, b, c, d (or d, c, b, a)!

**In general:** server learns order of records very quickly.

(If ages are in [0, N] and queries are uniform, the order is fully revealed after $O(N \log N)$ queries.)

If every value in [0, N] appears, position in order reveals value $\rightarrow$ encryption was useless!
Further motivation

Beside storage outsourcing: many situations where memory accesses of a sensitive algorithm can be observed by adversary (fully, or partially).

- **Trusted Enclaves** (e.g. Intel SGX): Client = CPU, Server = RAM. (Original motivation.)
- **Cache attacks** (concurrent processes observing cache misses).
- **Searchable Encryption**.

See: Side-channel attacks.
Oblivious RAM

Client stores $n$ data items.

“Key-value store”: each item is pair (address, data).

Client makes read/write operations:

- \texttt{read}(a): read data at address $a$; or
- \texttt{write}(a,d): write data $d$ at address $a$.

**Problem:** Client wants to do this without revealing which address is accessed.
**Oblivious RAM**

**ORAM algorithm C**: transforms each query $q$ by the client into one or several queries $C(q)$ to server.

**Obliviousness**: for any two sequences of queries $q = (q_1, \ldots, q_k)$ and $r = (r_1, \ldots, r_k)$ of the same length, $C(q) = (C(q_1), \ldots, C(q_k))$ and $C(r) = (C(r_1), \ldots, C(r_k))$ are indistinguishable.
**Trivial ORAM:** read and re-encrypt every item in server memory.

**Security:** trivial.

**Efficiency:** every client query costs $O(n)$ real accesses $\rightarrow$ overhead is $O(n)$.

A non-trivial ORAM must have:
- Client storage $o(n)$.
- Query overhead $o(n)$. 
Oblivious algorithm

ORAM wants to hide accesses made by the client, no matter what accesses the client is making.

This can be easier if the client is running a particular algorithm.

**Oblivious algorithm**: an algorithm $A$ is oblivious iff for any two inputs $x$ and $y$, the memory accesses of $A$ on input $x$, and $A$ on input $y$, are indistinguishable.
Roadmap

1. Oblivious Sorting.

2. General ORAM.
Oblivious Sorting
Oblivious algorithm: an algorithm $A$ is oblivious iff for any two inputs $x$ and $y$, the memory accesses of $A$ on input $x$, and $A$ on input $y$, are indistinguishable.

Which of the following algorithms are oblivious? (assuming inputs are arrays of fixed size.)

1. Bubble Sort. ✓ yes
2. Quick Sort. ✗ no
3. Merge Sort. ✗ no
Sorting obliviously

Basic operation: sorting two elements.

**Compare and swap:** on input \((x,y)\), if \(x < y\), output \((x,y)\), else output \((y,x)\).
Bubble Sort
**Sorting network**

**Comparator network:** A comparator network is an algorithm that consists in a sequence of compare-and-swaps ("comparators") between fixed inputs.

Can be represented in this form:

```
  +---+  +---+
 |   |  |   |
 +---+  +---+
```

**Sorting network:** A sorting network is a comparator network that correctly sorts its input (for all possible inputs).

*Remark:* testing whether a comparator network is a sorting network is co-NP-complete.
Size and depth

The **size** of a comparator network is its number of comparators.

The **depth** (or “critical path”) of a comparator network is the maximum number of comparators that an input value can go through. It is also the number of steps in a parallel computation of the network.

*Example*: comparator network of size 4 and depth 3.

→ Bubble sort is a sorting network of size $O(n^2)$ and depth $O(n)$. 
Can we do better?

**Proposition:** A sorting network must have size $\Omega(n \log n)$.

*Proof.* A network with $k$ comparators can permute its input sequence in at most $2^k$ different ways.

For a sorting network, we must have $2^k \geq n!$

By Stirling’s formula, this yields $k = \Omega(n \log n)$.

**Bitonic sort:** size $O(n \log^2 n)$.

Most efficient in practice.
The 0-1 principle

0-1 Principle: a comparator network (on \( n \) inputs) is a sorting network iff it correctly sorts all \( 2^n \) possible binary inputs.

Proof. Let \( f \) be a non-decreasing function: \( x \leq y \) implies \( f(x) \leq f(y) \).

Claim: If a comparator network has input \((x_1, \ldots, x_n)\) and outputs \((y_1, \ldots, y_n)\), then on input \((f(x_1), \ldots, f(x_n))\), it must output \((f(y_1), \ldots, f(y_n))\).

Proof of the claim: induction on comparators.

Now assume a comparator network is not a sorting network. Then there exist an input \((x_1, \ldots, x_n)\) and some indices \(i, j\), such that \(x_i < x_j\) but they are in the opposite order in the output.

Define \( f(x) = 0 \) if \( x \leq x_i \), 1 otherwise. We have \( f(x_i) = 0 \) and \( f(x_j) = 1 \), but their order is reversed by the network when inputting \((f(x_1), \ldots, f(x_n))\). Hence the network does not correctly sort all binary sequences.
Bitonic sequence: A sequence of values is bitonic iff:

- It is increasing, then decreasing.
- Or it is a circular shift of the previous case.

Example: bitonic sequences of 0 and 1’s are those of the form $0^a1^b0^c$ and $1^a0^b1^c$. 
Half-cleaner: A half-cleaner is a comparator network for an even number of inputs $n$, composed of comparators

$$(1,n/2+1), (2,n/2+2), \ldots, (n/2,n)$$

Half-cleaner for $n = 8$: 
**Key property of a half-cleaner:** if the input is bitonic, then both halves of the output are bitonic. Moreover, one of the two halves must be all 0’s or all 1’s. That half is called **clean**.
Bitonic sorter

Bitonic sorter recursive construction:

The bitonic sorter correctly sorts all bitonic inputs.
Batcher’s sort correctly sorts all binary inputs, hence all inputs.
Efficiency

A half-cleaner has size $H(n) = n/2$.

The bitonic sorter has size $S(n) = H(n) + 2S(n/2) = O(n \log n)$.

Batcher’s sort has size $B(n) = S(n) + 2B(n/2) = O(n \log^2 n)$.

The depth of Batcher’s sort is $O(\log^2 n)$: in a parallel computation model, only need $O(\log^2 n)$ steps.

→ Sorting algorithms used in GPUs.

Ajtai, Komlós, Szemerédi (STOC ’83): there exists a sorting network of size $O(n \log n)$.

Unfortunately, completely impractical.