Leakage in the Cell Probe Model
Lower Bounds for Response Hiding Encrypted Multi-Maps

Giuseppe Persiano

Università di Salerno

June, 2019

Describing joint work with:
Sarvar Patel and Kevin Yeo (Google LLC)
The Model

Cell Probe Model for a Data Structure [Yao]

- Memory is a sequence of *cells* each of *w* bits
- Accessing (reading/writing) a cell cost 1
- All computation is for free

Classical model used to derive lower bounds for Data Structures
The Oblivious Model

Oblivious Cell Probe Model [Larsen+Nielsen '18]

In a Client-Server setting
- Client outsources storage of the DS to an honest-but-curious server
- Client performs DS operations \( O = (\text{op}_1, \ldots, \text{op}_l) \) by accessing the Server memory
  - client can read and write any cell in Server memory
  - each cell is \( w \)-bit wide
- Client has limited private local memory
- Server observes the access pattern and the data downloaded
  - \( \text{view}^{\text{DS}}(O) = (\text{view}^{\text{DS}}(\text{op}_1), \ldots, \text{view}^{\text{DS}}(\text{op}_l)) \)
- Passive server: performs no computation
- Operations are performed online
Security Notion

**Definition**

**DS** is **Oblivious**, if for every PPT machine $A$ and any two sequences $O$ and $O'$ of the **same length**

$$\left| \text{Prob} \left[ A(\text{view}^{DS}(O)) = 1 \right] - \text{Prob} \left[ A(\text{view}^{DS}(O')) = 1 \right] \right| \leq \frac{1}{4}.$$
The array maintenance problem (a.k.a. ORAM)

Two operations to maintain an \( n \)-slot array \( A \)
- **Read\((i)\)** returns the current value stored in \( A[i] \)
- **Write\((i, x)\)** sets \( A[i] := x \)

**Theorem (Larsen+Nielsen '18)**

Expected amortized running time of an ORAM with \( n \) b-bit slots is \( \Omega \left( b w \cdot \log nb^c \right) \)

where \( c \) is the client memory in bits.
The array maintenance problem (a.k.a. ORAM)

Two operations to maintain an $n$-slot array $A$
- Read($i$) returns the current value stored in $A[i]$
- Write($i, x$) sets $A[i] := x$

**Theorem (Larsen+Nielsen ’18)**

Expected amortized running time of an ORAM with $n$ $b$-bit slots is

$$\Omega \left( \frac{b}{w} \cdot \log \frac{nb}{c} \right)$$

where $c$ is the client memory in bits.
The array maintenance problem (a.k.a. ORAM)

Two operations to maintain an \( n \)-slot array \( A \)
- \( \text{Read}(i) \) returns the current value stored in \( A[i] \)
- \( \text{Write}(i, x) \) sets \( A[i] := x \)

**Theorem (Larsen+Nielsen ’18)**

Expected amortized running time of an ORAM with \( n b \)-bit slots is

\[
\Omega \left( \frac{b}{w} \cdot \log \frac{nb}{c} \right)
\]

where \( c \) is the client memory in bits.

Online Read and Write operations with Passive Server
Proof strategy for ORAM lower bound [Larsen+Nielsen]

The Information Transfer Technique [Pătraşcu+Demaine]

- assign probes to nodes of the Information Tree
  - each probe to at most one node

- show that for most nodes $v$ there exists a hard distribution $\text{HD}_v$ on sequences of operations of the same length that assign lots of probes to $v$
  - coding argument leveraging on randomness of the entries of the array

- invoke obliviousness to show that for each such distribution all nodes must be assigned the same high number of probes
Obliviousness

- very strong requirement
- it hides the type of operation
- it hides the parameters of the operations
  - the content of the array (for Write)
  - the slot of the operation (for Read and Write)
- only number of operations is leaked
Obliviousness

- very strong requirement
- it hides the type of operation
- it hides the parameters of the operations
  - the content of the array (for Write)
  - the slot of the operation (for Read and Write)
- only number of operations is leaked

In several applications more information is leaked for the sake of efficiency
Differential Privacy

Definition

DS is $(\epsilon, \delta)$-DP, if for every PPT machine $A$ and any two sequences $O$ and $O'$ of the same length that differ for exactly one operation

$$\text{Prob} \left[ A(\text{view}^{\text{eMM}}(O)) = 1 \right] \leq e^\epsilon \cdot \text{Prob} \left[ A(\text{view}^{\text{eMM}}(O')) = 1 \right] + \delta$$
The Differentially Private RAM

Theorem (P+Yeo ’19)

For every $\epsilon > 0$ and $\delta \leq 1/3$, the expected amortized running time of a Differentially Private RAM with $n$ $b$-bit slots is

$$\Omega \left( \frac{b}{w} \cdot \log \frac{nb}{c} \right)$$

where $c$ is the client memory in bits.
The Differentially Private RAM

Theorem (P+Yeo ’19)

For every $\epsilon > 0$ and $\delta \leq 1/3$, the expected amortized running time of a Differentially Private RAM with $n$ $b$-bit slots is

$$\Omega \left( \frac{b}{w} \cdot \log \frac{nb}{c} \right)$$

where $c$ is the client memory in bits.

Different proof technique
A sequence of operations $O = (\text{op}_1, \text{op}_2, \ldots, \text{op}_l)$ is associated with leakage $\mathcal{L}(O)$:

$$\mathcal{L}(O) = (\mathcal{L}(\text{op}_1), \ldots, \mathcal{L}(\text{op}_l))$$
Leakage Cell Probe Model

A sequence of operations \( O = (\text{op}_1, \text{op}_2, \ldots, \text{op}_l) \) is associated with leakage \( \mathcal{L}(O) \)

\[
\mathcal{L}(O) = (\mathcal{L}(\text{op}_1), \ldots, \mathcal{L}(\text{op}_l))
\]

**Definition**

**DS** is Non-Adaptively \( \mathcal{L} \)-INDSecure, if for every PPT machine \( A \) and any two sequences \( O \) and \( O' \) such that \( \mathcal{L}(O) = \mathcal{L}(O') \),

\[
\left| \text{Prob} \left[ A(\text{view}^{\text{DS}}(O)) = 1 \right] - \text{Prob} \left[ A(\text{view}^{\text{DS}}(O')) = 1 \right] \right| \leq \frac{1}{4}.
\]
Leakage Cell Probe Model

A sequence of operations \( O = (\text{op}_1, \text{op}_2, \ldots, \text{op}_l) \) is associated with leakage \( \mathcal{L}(O) \)

\[
\mathcal{L}(O) = (\mathcal{L}(\text{op}_1), \ldots, \mathcal{L}(\text{op}_l))
\]

Definition

DS is Non-Adaptively \( \mathcal{L} \)-INDSecure, if for every PPT machine \( \mathcal{A} \) and any two sequences \( O \) and \( O' \) such that \( \mathcal{L}(O) = \mathcal{L}(O') \),

\[
\left| \text{Prob} \left[ \mathcal{A}(\text{view}^{\text{DS}}(O)) = 1 \right] - \text{Prob} \left[ \mathcal{A}(\text{view}^{\text{DS}}(O')) = 1 \right] \right| \leq \frac{1}{4}.
\]

Oblivious considers leakage \( \mathcal{L}(O) = I \)
Multi-Maps (MM)

Multi-Maps

A **data structure** to maintain a collection of pairs \((\text{key}, \vec{v})\), where \(\vec{v} = (v_1, \ldots, v_l)\) is a tuple

1. **Add(key, v)**: *adds v to the tuple associated with key*

2. **Get(key)**: *returns the tuple associated with key*
Multi-Maps (MM)

Multi-Maps

A data structure to maintain a collection of pairs \((\text{key}, \vec{v})\), where \(\vec{v} = (v_1, \ldots, v_l)\) is a tuple

1. **Add(key, v):** adds \(v\) to the tuple associated with \(\text{key}\)

2. **Get(key):** returns the tuple associated with \(\text{key}\)

- A special case of **Structured Encryption** [Chase-Kamara]
Multi-Maps (MM)

Multi-Maps

A data structure to maintain a collection of pairs \((\text{key}, \vec{v})\), where \(\vec{v} = (v_1, \ldots, v_l)\) is a tuple

1. Add(\text{key}, v): adds \(v\) to the tuple associated with \text{key}

2. Get(\text{key}): returns the tuple associated with \text{key}

- A special case of Structured Encryption [Chase-Kamara]
- A generalization of ORAM:
Multi-Maps (MM)

Multi-Maps

A data structure to maintain a collection of pairs \((\text{key}, \vec{v})\), where \(\vec{v} = (v_1, \ldots, v_l)\) is a tuple

1. Add\((\text{key}, v)\): adds \(v\) to the tuple associated with \(\text{key}\)

2. Get\((\text{key})\): returns the tuple associated with \(\text{key}\)

- A special case of \textit{Structured Encryption} [Chase-Kamara]
- A generalization of ORAM:
  - ORAM is a MM with all tuples of length 1;
How expensive are EMM?

It depends on the leakage function. If no security is sought:

\[ O(\log \log n \log \log \log n) \] \[\text{[Beame and Fich '99]}\]

If only the number of operations is leaked:

\[ O(\log n) \]

Use ORAM [Folklore].

What if we only want to hide the response of the operations? What is the cost of the Response-Hiding EMM?
How expensive are EMM?

It depends on the leakage function

\[ O(\log \log n \log \log \log n) \] 
\[ \text{Beame and Fich '99} \]

If only number of operations is leaked
\[ O(\log n) \]
Use ORAM [Folklore]

What if we only want to hide the response of the operations?

What is the cost of the Response-Hiding EMM?
How expensive are EMM?

It depends on the leakage function

If no security is sought:

$$O\left(\frac{\log \log n}{\log \log \log n}\right)$$

[Beame and Fich '99]
How expensive are EMM?

It depends on the leakage function

If no security is sought:

\[ O \left( \frac{\log \log n}{\log \log \log n} \right) \]

[Beame and Fich ’99]

If only number of operations is leaked

\[ O (\log n) \]

Use ORAM [Folklore]
How expensive are EMM?

It depends on the leakage function

If no security is sought:

$$O \left( \frac{\log \log n}{\log \log \log n} \right)$$

[Beame and Fich '99]

If only number of operations is leaked

$$O (\log n)$$

Use ORAM [Folklore]

What if we only want to hide the response of the operations?
How expensive are EMM?

It depends on the leakage function

If no security is sought:

\[ O \left( \frac{\log \log n}{\log \log \log n} \right) \]

[Beame and Fich '99]

If only number of operations is leaked

\[ O(\log n) \]

Use ORAM [Folklore]

What if we only want to hide the response of the operations?

What is the cost of the Response-Hiding EMM?
Response-Hiding Leakage Function – I

Definition (Leakage function $\mathcal{L}^G$ for $O = (\text{op}_1, \ldots, \text{op}_l)$)

$\mathcal{L}^G(O_i)$ is defined as follows:

1. if $\text{op}_i = \text{Get}(\text{key}_i)$ then $\mathcal{L}^G(O_i) = (\text{Get}, \text{key}_i, |\text{Get (MM}^{O_{i-1}}, \text{key}_i)|)$;
   - the key queried and the size of the response are leaked

2. if $\text{op}_i = \text{Add}(\text{key}_i, v_i)$ then $\mathcal{L}^G(O_i) = (\text{Add}, \text{aep}^i)$
   - the add pattern is leaked
   - the type of operation is also leaked

*add equality pattern* $\text{aep}^i := (\text{aep}^i_1, \ldots, \text{aep}^i_{i-1})$ and $\text{aep}^i_j$ is defined as follows, for $j = 1, \ldots, i - 1$

$$\text{aep}^i_j = \begin{cases} 
\bot, & \text{if } \text{op}_j \text{ is a Get operation;} \\
0, & \text{if } \text{op}_j \text{ is an Add operation and } \text{key}_j \neq \text{key}_i; \\
1, & \text{if } \text{op}_j \text{ is an Add operation and } \text{key}_j = \text{key}_i;
\end{cases}$$
Response-Hiding Leakage Function – II

**Definition (Leakage function \( \mathcal{L}^A \) for \( O = (\text{op}_1, \ldots, \text{op}_i) \))**

\( \mathcal{L}^A(O_i) \) is defined as follows:

1. If \( \text{op}_i = \text{Get}(\text{key}_i) \) then \( \mathcal{L}^A(O_i) = (\text{Get}, |\text{Get}(\text{MM}^{O_{i-1}}, \text{key}_i)|, \text{gep}_i) \);
   the size of the response and the equality pattern are leaked.

2. If \( \text{op}_i = \text{Add}(\text{key}_i, v_i) \) then \( \mathcal{L}^A(O_i) = (\text{Add}, \text{key}_i, v_i) \);
   all the parameters of an Add
   the type of operation is also leaked.

*get equality pattern* \( \text{gep}_i := (\text{gep}_i^1, \ldots, \text{gep}_i^{i-1}) \) and \( \text{gep}_j^i \) is defined as follows, for \( j = 1, \ldots, i - 1 \):

\[
\text{gep}_j^i = \begin{cases} 
\bot, & \text{if } \text{op}_j \text{ is a Add operation;} \\
0, & \text{if } \text{op}_j \text{ is an Get operation and } \text{key}_j \neq \text{key}_i; \\
1, & \text{if } \text{op}_j \text{ is an Get operation and } \text{key}_j = \text{key}_i;
\end{cases}
\]
Main result

Theorem (Informal)

$L^G$-INDSecurity and $L^A$-INDSecurity EMM have $\Omega(\log n)$ expected amortized overhead.
Main result

Theorem (Informal)

$L^G$-INDSecurity and $L^A$-INDSecurity EMM have $\Omega(\log n)$ expected amortized overhead.

A sequence of operations that return $R$ responses requires $\Omega(R \cdot \log n)$ work.
Main result

Theorem (Informal)

\( \mathcal{L}^G \)-INDSecurity and \( \mathcal{L}^A \)-INDSecurity EMM have \( \Omega(\log n) \) expected amortized overhead.

A sequence of operations that return \( R \) responses requires \( \Omega(R \cdot \log n) \) work.

This is tight [Folklore]

- Use ORAM and spend \( O(\log n) \)
Proof technique

We adapt the Information Transfer technique of [P+D] to our setting

- we have a **weaker** security notion
  - can only invoke obliviousness for distribution with same leakage
  - we prove lower bound for very **leaky** implementations

- in our data structure problem entries/values are **not** random
  - need to identify a different source of randomness for the encoding argument
Defining the Hard Distribution $\text{HD}$ for $L^G$

we have

1. the following disjoint sets of values
   - $V_0$ consisting of $k$ values;
   - $V_1, \ldots, V_p$ each consisting of $n^\epsilon$ values;

2. the following disjoint sets of keys:
   - sets $K_i^a$, for $i = 1, \ldots, p$, each of size $n^\epsilon$;
   - sets $K_i^g$, for $i = 1, \ldots, p$, each of size $n^\epsilon$;
Defining the Hard Distribution \( \text{HD} \) for \( \mathcal{L}^G \)

we have

1. the following disjoint sets of values
   - \( V_0 \) consisting of \( k \) values;
   - \( V_1, \ldots, V_p \) each consisting of \( n^\epsilon \) values;

2. the following disjoint sets of keys:
   - sets \( K^a_i \), for \( i = 1, \ldots, p \), each of size \( n^\epsilon \);
   - sets \( K^g_i \), for \( i = 1, \ldots, p \), each of size \( n^\epsilon \);

\[
p = n^{1-\epsilon}
\]
Defining the Hard Distribution **HD**

**Phase 0**

Execute **SubPhase I**\(_i\), for \(i = 1, \ldots, p\)

for each \(\text{key} \in K_i^g\)

**output:** \(\text{Add(key, } V_0)\),

**Phase \(j\), for \(j = 1, \ldots, p\)**

Execute **SubPhase A**\(_j\) and **SubPhase G**\(_j\)

- **SubPhase A**\(_j\)
  for each \(\text{key} \in K_j^a\),
  randomly select subset \(B_{\text{key}} \subset V_j\) of \(k\) values
  **output:** \(\text{Add(key, } B_{\text{key}})\);

- **SubPhase G**\(_j\)
  for each \(\text{key} \in K_j^g\)
  **output:** \(\text{Get(key)}\);
The Hard Distribution HD

InitPhase

\[
\begin{align*}
\text{Add}(K^g_1, V_0) & \quad \text{Get}(K^g_1) \\
\text{Add}(K^g_2, V_0) & \quad \text{Get}(K^g_2) \\
\vdots & \quad \vdots \\
\text{Add}(K^g_j, V_0) & \quad \text{Get}(K^g_j) \\
\text{Add}(K^g_p, V_0) & \quad \text{Get}(K^g_p)
\end{align*}
\]
The Hard Distribution \textbf{HD}

\textbf{InitPhase}

\begin{align*}
I_1 & \quad I_2 \\
\text{Add}(K_1^g, V_0) & \quad \text{Add}(K_2^g, V_0) \\
\text{same key} & \quad \text{same key}
\end{align*}

\begin{align*}
I_j & \quad I_p \\
\text{Add}(K_j^g, V_0) & \quad \text{Add}(K_p^g, V_0) \\
\text{same key} & \quad \text{same key}
\end{align*}

\begin{align*}
A_1 & \quad G_1 \\
\text{Add}(K_1^a) & \quad \text{Get}(K_1^g)
\end{align*}

\begin{align*}
A_i & \quad G_i \\
\text{Add}(K_i^a) & \quad \text{Get}(K_i^g)
\end{align*}

\begin{align*}
A_p & \quad G_p \\
\text{Add}(K_p^a) & \quad \text{Get}(K_p^g)
\end{align*}
The Hard Distribution **HD**

InitPhase

\[
\begin{align*}
& \text{Add}(K^g_1, V_0) \quad \text{same key} \\
& \text{Add}(K^g_2, V_0) \quad \text{same key} \\
& \text{Add}(K^g_j, V_0) \quad \text{same key} \\
& \text{Add}(K^g_p, V_0) \quad \text{same key}
\end{align*}
\]

\[
\begin{align*}
& \text{Add}(K^a_1) \quad \text{same key} \\
& \text{Get}(K^g_1) \\
& \text{Add}(K^a_i) \quad \text{same key} \\
& \text{Get}(K^g_i) \\
& \text{Add}(K^a_p) \quad \text{same key} \\
& \text{Get}(K^g_p)
\end{align*}
\]
The Hard Distribution \textbf{HD} \\

\textbf{InitPhase} \\

\begin{align*}
&I_1 \quad I_2 \quad \ldots \quad I_j \quad \ldots \quad I_p \\
&\text{Add}(K_1^g, V_0) \quad \text{Add}(K_2^g, V_0) \quad \text{Add}(K_j^g, V_0) \quad \text{Add}(K_p^g, V_0) \\
&\text{same key} \quad \text{same key} \quad \text{same key} \quad \text{same key}
\end{align*}

\begin{align*}
&A_1 \quad G_1 \quad \ldots \quad A_i \quad \ldots \quad A_p \\
&\text{Add}(K_1^a) \quad \text{Get}(K_1^g) \quad \text{Add}(K_i^a) \quad \text{Get}(K_i^g) \quad \text{Add}(K_p^a) \quad \text{Get}(K_p^g) \\
&\text{same key} \quad K_1^g, V_0 \quad \text{same key} \quad K_i^g, V_0 \quad \text{same key} \quad K_p^g, V_0
\end{align*}
The Hard Distribution **HD**

**InitPhase**

- **Add($K^g_1, V_0$)**: same key
- **Add($K^g_2, V_0$)**: same key
- **Add($K^g_j, V_0$)**: same key
- **Add($K^g_p, V_0$)**: same key

- **Add($K^a_1$)**: same key
- **Get($K^g_1$)**: $K^g_1, k$
- **Add($K^a_i$)**: same key
- **Get($K^g_i$)**: $K^g_i, k$
- **Add($K^a_p$)**: same key
- **Get($K^g_p$)**: $K^g_p, k$
The Information Tree of the Hard Distribution

Each probe is assigned to at most one node.

Giuseppe Persiano (UNISA)
June 2019
Each probe is assigned to at most one node.
Each probe is assigned to at most one node.

write(21,...)
The Information Tree of the Hard Distribution

Each probe is assigned to at most one node.
Each probe is assigned to at most one node.
Each probe is assigned to at most one node.
Each probe is assigned to at most one node.
Each probe is assigned to at most one node.
Each probe is assigned to at most one node

write(21,...)
Each probe is assigned to at most one node
Each probe is assigned to at most one node.

write\((16,\ldots)\)
Each probe is assigned to at most one node.
Each probe is assigned to at most one node.
Each probe is assigned to at most one node.
The Information Tree of the Hard Distribution

Each probe is assigned to at most one node
Each probe is assigned to at most one node
The Neighbor Hard Distributions

InitPhase

\[ \text{Add}(K_{1}^{g}, V_{0}) \rightarrow \text{Add}(K_{2}^{g}, V_{0}) \rightarrow \ldots \rightarrow \text{Add}(K_{j}^{g}, V_{0}) \rightarrow \ldots \rightarrow \text{Add}(K_{p}^{g}, V_{0}) \]

\[ \text{Add}(K_{1}^{a}) \rightarrow \text{Get}(K_{1}^{g}) \rightarrow \text{Add}(K_{i}^{a}) \rightarrow \text{Get}(K_{i}^{g}) \rightarrow \text{Add}(K_{p}^{a}) \rightarrow \text{Get}(K_{p}^{g}) \]
The Neighbor Hard Distributions

InitPhase

\[ I_1 \quad I_2 \quad \ldots \quad I_j \quad I_p \]

Add\((K_1, V_0)\) \quad Add\((K_2, V_0)\) \quad Add\((K_i^a)\) \quad Add\((K_p, V_0)\)

\[ A_1 \quad G_1 \quad \ldots \quad A_i \quad G_i \quad \ldots \quad A_p \quad G_p \]

Add\((K_1^a)\) \quad Get\((K_1^g)\) \quad Add\((K_j, V_0)\) \quad Get\((K_i^g)\) \quad Add\((K_p^a)\) \quad Get\((K_p^g)\)
The Neighbor Hard Distributions

InitPhase

\[
\begin{align*}
I_1 & \longrightarrow \text{Add}(K_1^g, V_0) \\
I_2 & \longrightarrow \text{Add}(K_2^g, V_0)
\end{align*}
\]

\[
\begin{align*}
I_j & \longrightarrow \text{Add}(K_i^a) \\
I_p & \longrightarrow \text{Add}(K_p^g, V_0)
\end{align*}
\]

\[
\begin{align*}
A_1 & \longrightarrow \text{Add}(K_1^a) \\
G_1 & \longrightarrow \text{Get}(K_1^g)
\end{align*}
\]

\[
\begin{align*}
A_i & \longrightarrow \text{Add}(K_i^g, V_0) \\
G_i & \longrightarrow \text{Get}(K_i^g)
\end{align*}
\]

\[
\begin{align*}
A_p & \longrightarrow \text{Add}(K_p^a) \\
G_p & \longrightarrow \text{Get}(K_p^g)
\end{align*}
\]

same key

Giuseppe Persiano (UNISA)
June 2019 21 / 27
The Neighbor Hard Distributions

InitPhase

\[ \text{Add}(K_{g_1}^g, V_0) \]
\[ \text{Add}(K_{g_2}^g, V_0) \]
\[ \text{Add}(K_{g_i}^a) \]
\[ \text{Add}(K_{g_p}^g, V_0) \]

\[ \text{same key} \]
\[ \text{same key} \]
\[ \text{same key} \]
\[ \text{same key} \]

\[ A_1 \]
\[ G_1 \]
\[ \text{Add}(K_{a_1}^a) \]
\[ \text{Get}(K_{g_1}^g) \]
\[ K_{g_1}, k \]

\[ A_i \]
\[ G_i \]
\[ \text{Add}(K_{a_i}^a) \]
\[ \text{Get}(K_{g_i}^g) \]
\[ K_{g_i}, k \]

\[ A_p \]
\[ G_p \]
\[ \text{Add}(K_{a_p}^a) \]
\[ \text{Get}(K_{g_p}^g) \]
\[ K_{g_p}, k \]

Giuseppe Persiano (UNISA)

June 2019 21 / 27
$\text{HD}_v$: Hard distribution at $v$
$\textbf{HD}_v$: Hard distribution at $v$
Get operations in the right subtree

Add operations in the left subtree
Get operations in the right subtree

Client memory

Cells overwritten in right subtree

Add operations in the left subtree
Get operations in the right subtree

Client memory

Cells overwritten in right subtree

Add operations in the left subtree

Each keyword receives $k$ random values from a set of $n^\epsilon$
Get operations in the right subtree

Client memory

Cells overwritten in right subtree

Entropy:
\[ \log \left( \binom{n^c}{k} \right) \Omega(k \log n) \text{ bits} \]

Add operations in the left subtree
Theorem

For every $v$ of the information tree of depth $8 \leq d \leq \frac{1-\epsilon}{2} \log \frac{n}{c}$

$$\mathbb{E} [\left| \text{Count}(v) \right|] = \Omega \left( \frac{n}{2^d} \cdot k \cdot \frac{\log n}{w} \right)$$

with respect to $\text{HD}_v$. 
Theorem

For every $v$ of the information tree of depth $8 \leq d \leq \frac{1-\epsilon}{2} \log \frac{n}{c}$

$$E[|\text{Count}(v)|] = \Omega \left( \frac{n}{2^d} \cdot k \cdot \frac{\log n}{w} \right)$$

with respect to $\text{HD}_v$.

For every $v$, $\mathcal{L}^G(\text{HD}_v) = \mathcal{L}^G(\text{HD})$, so by $\mathcal{L}^G$-INDsecurity,
Theorem

For every $v$ of the information tree of depth $8 \leq d \leq \frac{1-\epsilon}{2} \log \frac{n}{c}$

$$\mathbb{E} [|\text{Count}(v)|] = \Omega \left( \frac{n}{2^d} \cdot k \cdot \frac{\log n}{w} \right)$$

with respect to $\text{HD}_v$.

For every $v$, $L^G(\text{HD}_v) = L^G(\text{HD})$, so by $L^G$-INDsecurity,

Theorem

For every $v$ of the information tree of depth $8 \leq d \leq \frac{1-\epsilon}{2} \log \frac{n}{c}$

$$\mathbb{E} [|\text{Count}(v)|] = \Omega \left( \frac{n}{2^d} \cdot k \cdot \frac{\log n}{w} \right)$$

with respect to $\text{HD}$. 
Wrapping up

For an eMM that is $\mathcal{L}^G$-IND secure

- each probe contributes 1 to at most one $\text{Count}(v)$. 
Wrapping up

For an eMM that is $\mathcal{L}^G$-IND secure

- each probe contributes 1 to at most one $\text{Count}(v)$.
  - $\sum_v \text{Count}(v)$ is a lower bound to the number of probes
Wrapping up

For an eMM that is $\mathcal{L}^G$-IND secure

- each probe contributes 1 to at most one $\text{Count}(v)$.
  - $\sum_v \text{Count}(v)$ is a lower bound to the number of probes
- level $d$ has $2^d$ nodes,
Wrapping up

For an eMM that is $\mathcal{L}^G$-IND secure

- each probe contributes 1 to at most one $\text{Count}(v)$.
  - $\sum_v \text{Count}(v)$ is a lower bound to the number of probes

- level $d$ has $2^d$ nodes,
  - each level contributes $n \cdot k \cdot \frac{\log n}{w}$

$\Theta(nk)$ Add
$\Theta(n)$ Get each with $\Theta(k)$ results each

$\Omega(\log n \log n^c)$ amortized efficiency per response
Wrapping up

For an eMM that is $\mathcal{L}^G$-IND secure

- each probe contributes 1 to at most one $\text{Count}(v)$.
  - $\sum_v \text{Count}(v)$ is a lower bound to the number of probes
- level $d$ has $2^d$ nodes,
  - each level contributes $n \cdot k \cdot \frac{\log n}{w}$
- we have $\Theta(\log \frac{n}{c})$ levels
Wrapping up

For an eMM that is $\mathcal{L}^G$-IND secure

- each probe contributes 1 to at most one $\text{Count}(v)$.
  - $\sum_v \text{Count}(v)$ is a lower bound to the number of probes
- level $d$ has $2^d$ nodes,
  - each level contributes $n \cdot k \cdot \frac{\log n}{w}$
- we have $\Theta(\log \frac{n}{c})$ levels
- number of probes is

$$\Omega \left( n \cdot k \cdot \frac{\log n}{w} \cdot \log \frac{n}{c} \right)$$

...to execute
Wrapping up

For an eMM that is $L^G$-IND secure

- each probe contributes 1 to at most one $\text{Count}(v)$.
  - $\sum_v \text{Count}(v)$ is a lower bound to the number of probes
- level $d$ has $2^d$ nodes,
  - each level contributes $n \cdot k \cdot \frac{\log n}{w}$
- we have $\Theta(\log \frac{n}{c})$ levels
- number of probes is

$$\Omega \left( n \cdot k \cdot \frac{\log n}{w} \cdot \log \frac{n}{c} \right)$$

to execute
  - $\Theta(nk)$ Add
Wrapping up

For an eMM that is $\mathcal{L}^G$-IND secure

- each probe contributes 1 to at most one $\text{Count}(v)$.
  - $\sum_v \text{Count}(v)$ is a lower bound to the number of probes
- level $d$ has $2^d$ nodes,
  - each level contributes $n \cdot k \cdot \frac{\log n}{w}$
- we have $\Theta(\log \frac{n}{c})$ levels
- number of probes is

$$\Omega \left( n \cdot k \cdot \frac{\log n}{w} \cdot \log \frac{n}{c} \right)$$

to execute
  - $\Theta(nk)$ Add
  - $\Theta(n)$ Get each with $\Theta(k)$ results each
Wrapping up

For an eMM that is $\mathcal{L}^G$-IND secure

- each probe contributes 1 to at most one $\text{Count}(v)$.
  - $\sum_v \text{Count}(v)$ is a lower bound to the number of probes
- level $d$ has $2^d$ nodes,
  - each level contributes $n \cdot k \cdot \frac{\log n}{w}$
- we have $\Theta(\log \frac{n}{c})$ levels
- number of probes is

$$\Omega\left(n \cdot k \cdot \frac{\log n}{w} \cdot \log \frac{n}{c}\right)$$

to execute
  - $\Theta(nk)$ Add
  - $\Theta(n)$ Get each with $\Theta(k)$ results each

- amortized efficiency per response

$$\Omega\left(\frac{\log n}{w} \cdot \log \frac{n}{c}\right)$$
Typical parameter regime

\[ w = \Omega(\log n) \text{ and } c = n^\alpha, \alpha < 1. \]
Typical parameter regime

\[ w = \Omega(\log n) \text{ and } c = n^\alpha, \alpha < 1. \]

amortized efficiency per response of an eMM is

\[ \Omega(\log n) \]
Typical parameter regime

\[ w = \Omega(\log n) \text{ and } c = n^\alpha, \quad \alpha < 1. \]

amortized efficiency per response of an eMM is

\[ \Omega (\log n) \]

Same for \( \mathcal{L}^A \) leakage function
Conclusions

- Response Hiding in a *mildly* Dynamic setting gives $\Omega(\log n)$ overhead
  - static EMM can be implemented with *constant* slowdown via cuckoo hashing
Conclusions

- Response Hiding in a *mildly* Dynamic setting gives $\Omega(\log n)$ overhead
  - static EMM can be implemented with constant slowdown via cuckoo hashing
  - proof only uses addition of values to keys
Conclusions

- Response Hiding in a *mildly* Dynamic setting gives $\Omega(\log n)$ overhead
  - static EMM can be implemented with constant slowdown via cuckoo hashing
  - proof only uses addition of values to keys
  - no remove operation