The Curse of the Length
The Case of Encrypted Multi-Maps

Giuseppe Persiano

Università di Salerno

August, 2020

Mitigating Leakage in Secure Cloud-Hosted Data Structures: Volume-Hiding for Multi-Maps via Hashing
by Sarvar Patel, GP, Kevin Yeo, and Moti Yung
CCS '19

Start from the beginning

Reprinted from JOURNAL OF COMPUTER AND SYSTEM SCIENCES

Vol. 28, No. 2, April 1984
Printed in Belgium

Probabilistic Encryption*

SHAFI GOLDWASSER AND SILVIO MICALI

Laboratory of Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139

Received February 3, 1983; revised November 8, 1983

Definition of Secure Encryption

Reprinted from Journal of Computer and System Sciences

Vol. 28, No. 2, April 1984
Printed in Belgium

Probabilistic Encryption*

Shafi Goldwasser and Silvio Micali

Laboratory of Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139

Received February 3, 1983; revised November 8, 1983

A new probabilistic model of data encryption is introduced. For this model, under suitable complexity assumptions, it is proved that extracting *any information* about the cleartext from the cyphertext is hard on the average for an adversary with polynomially bounded computational resources. The proof holds for any message space with any probability distribution. The first implementation of this model is presented. The security of this implementation is proved under the intractability assumption of deciding Quadratic Residuosity modulo composite numbers whose factorization is unknown.

1. Introduction

This paper proposes an encryption scheme that possesses the following property:

Whatever is efficiently computable about the cleartext given the cyphertext, is also efficiently computable without the cyphertext.

The fine print

Formal Setting

Let Π be a PKC. Let MG be a message generator. We write M_k for MG$[k]$. Without loss of generality, we assume that all $m \in M_k$ have the same length $l_k = Q(k)$ for some polynomial Q.

The fine print

Formal Setting

Let Π be a PKC. Let MG be a message generator. We write M_k for MG$[k]$. Without loss of generality, we assume that all $m \in M_k$ have the same length $l_k = Q(k)$ for some polynomial Q.

Indeed WLOG:

The fine print

Formal Setting

Let Π be a PKC. Let MG be a message generator. We write M_k for MG$[k]$. Without loss of generality, we assume that all $m \in M_k$ have the same length $l_k = Q(k)$ for some polynomial Q.

Indeed WLOG:

Just pad each message in the message space to the maximum length

The fine print

Formal Setting

Let Π be a PKC. Let MG be a message generator. We write M_k for MG$[k]$. Without loss of generality, we assume that all $m \in M_k$ have the same length $l_k = Q(k)$ for some polynomial Q.

Encryption necessarily leaks an upper bound on the length of the plaintext

Incompressibility

Fact of life

Encryption necessarily leaks an upper bound on the length of the plaintext

A direct consequence of Shannon/Kolmogoroff

Structured Encryption

Chase-Kamara 2010

- Data is often organized in Data Structures

Structured Encryption

Chase-Kamara 2010

- Data is often organized in Data Structures
- For efficient retrieval

Structured Encryption

Chase-Kamara 2010

- Data is often organized in Data Structures
- For efficient retrieval
- Storage is outsourced to untrusted server

Structured Encryption   Chase-Kamara 2010

- Data is often organized in Data Structures
- For efficient retrieval
- Storage is outsourced to untrusted server
  - honest but very curious

(Plaintext) Multi-Maps

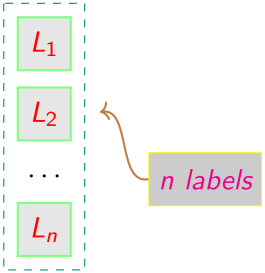
L_1

L_2

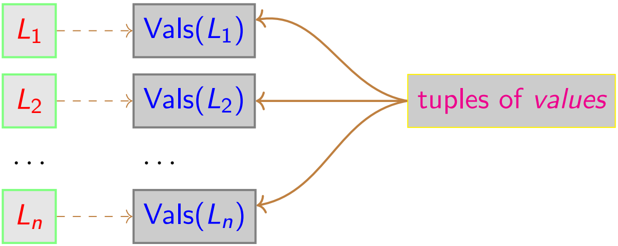
$\cdots$

L_n

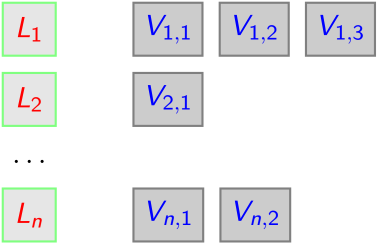
(Plaintext) Multi-Maps



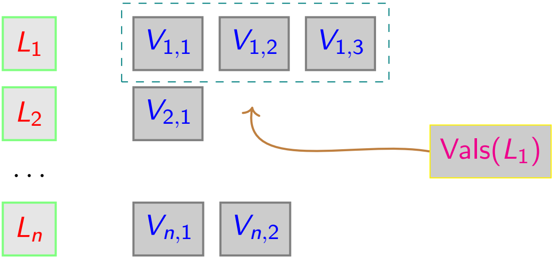
(Plaintext) Multi-Maps



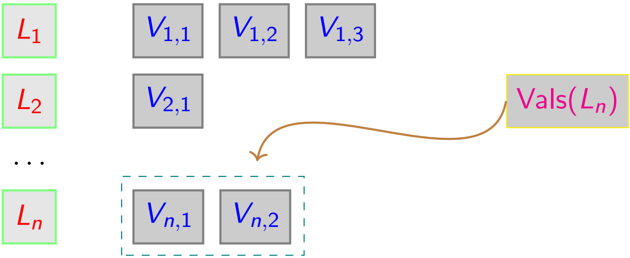
(Plaintext) Multi-Maps



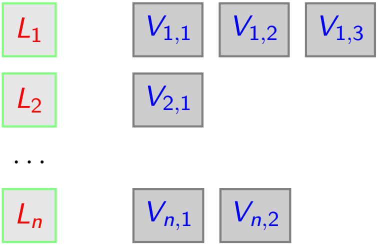
(Plaintext) Multi-Maps



(Plaintext) Multi-Maps



(Plaintext) Multi-Maps

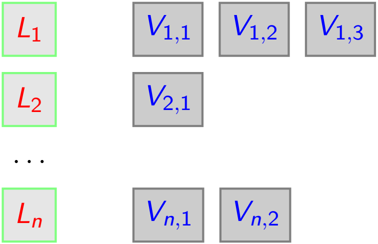


Supported operations

$\mathsf{Init}((L_i, \mathsf{Vals}(L_i))_i)$

$\mathsf{Get}(L) \rightarrow \mathsf{Vals}(L)$

(Plaintext) Multi-Maps



Supported operations

$\mathsf{Init}((L_i, \mathsf{Vals}(L_i))_i)$

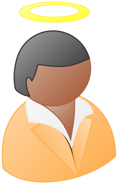
$\mathsf{Get}(L) \to \mathsf{Vals}(L)$

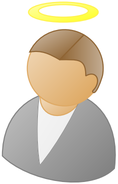
Inverted index

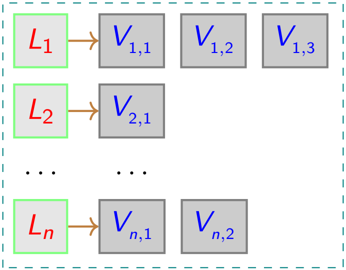
Labels are keywords

Values are documents

Plaintext Multi-Maps



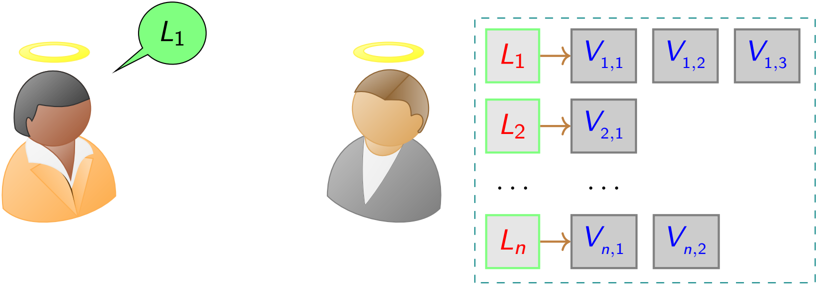




User

Cloud Manager

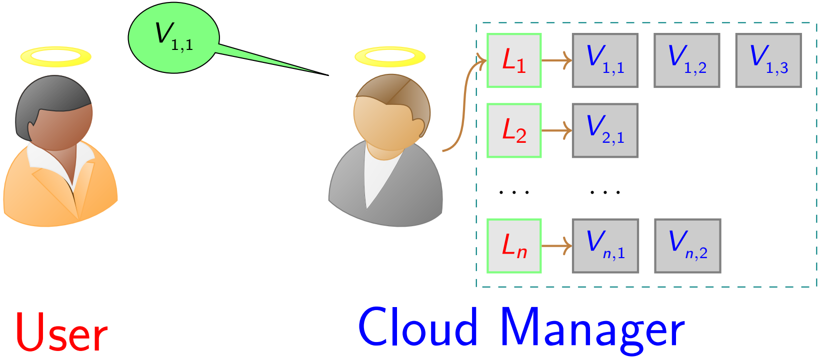
Plaintext Multi-Maps



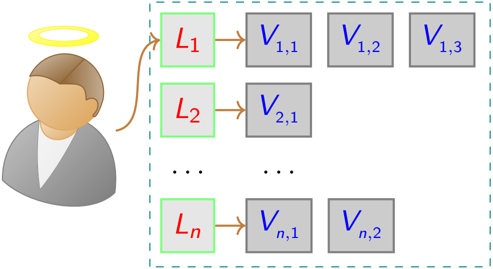
User

Cloud Manager

Plaintext Multi-Maps



Plaintext Multi-Maps

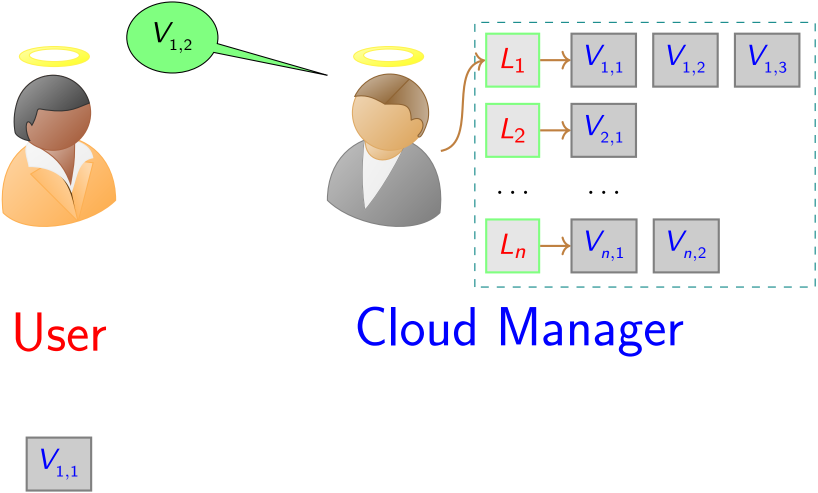


User

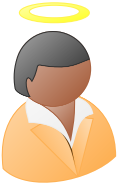
Cloud Manager

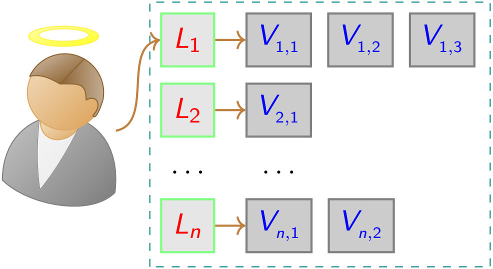
$V_{1,1}$

Plaintext Multi-Maps



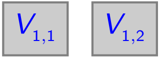
Plaintext Multi-Maps



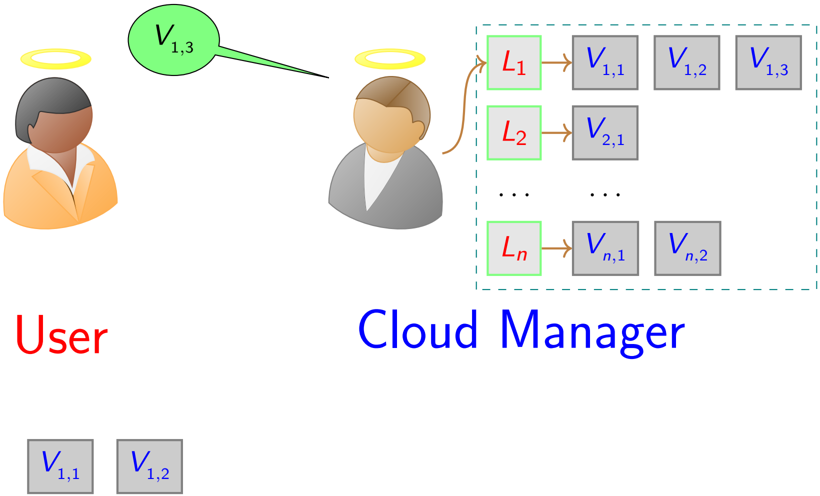


User

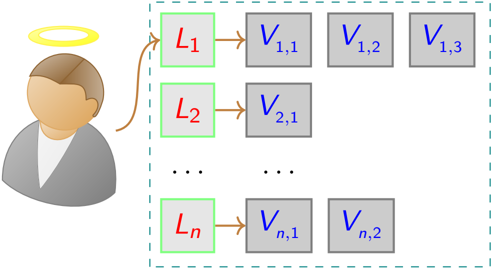
Cloud Manager



Plaintext Multi-Maps

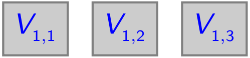


Plaintext Multi-Maps

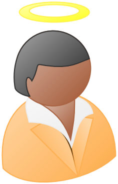


User

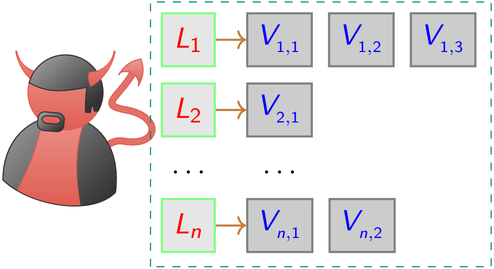
Cloud Manager



Plaintext Multi-Maps with Evil Cloud Manager

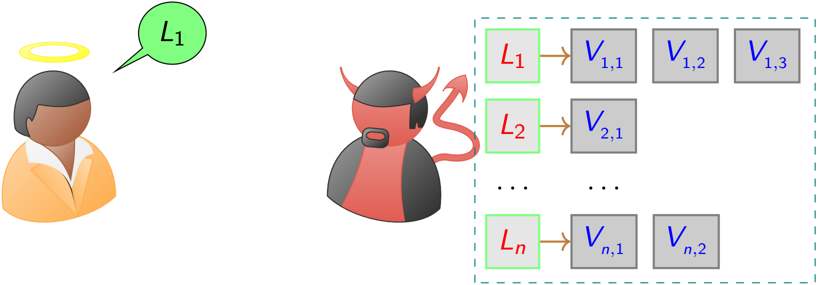


User



Cloud Manager

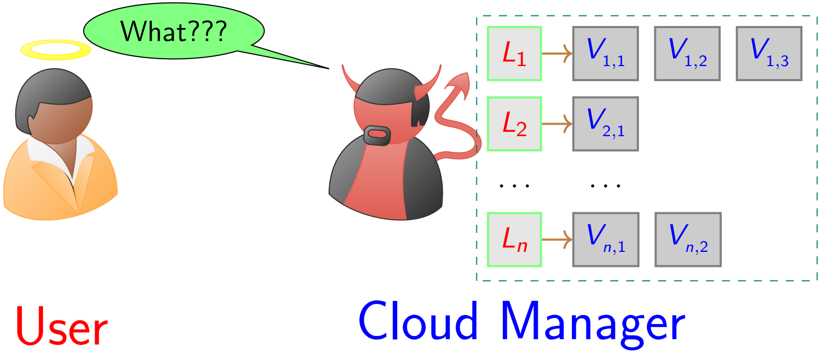
Plaintext Multi-Maps with Evil Cloud Manager



User

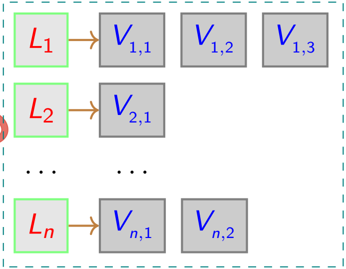
Cloud Manager

Plaintext Multi-Maps with Evil Cloud Manager



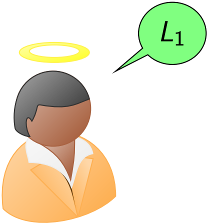
Plaintext Multi-Maps with Honest-but-Curious Cloud Manager

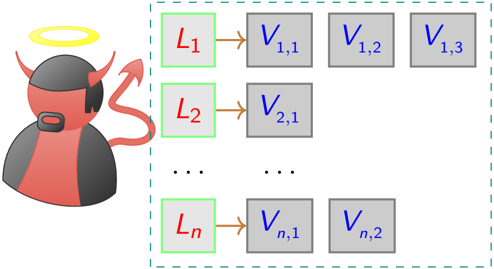
User



Cloud Manager

Plaintext Multi-Maps with Honest-but-Curious Cloud Manager

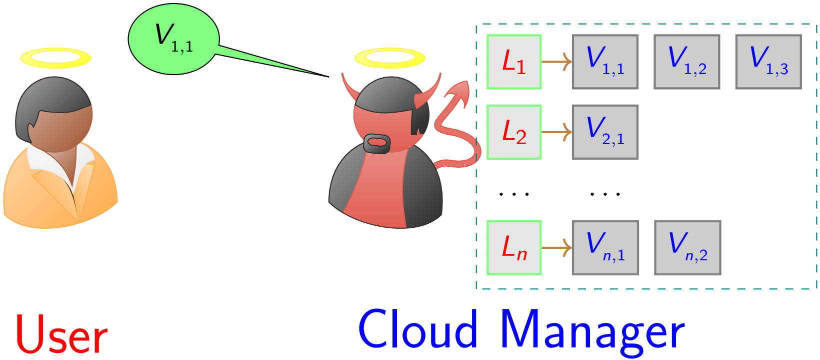




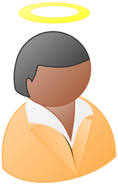
User

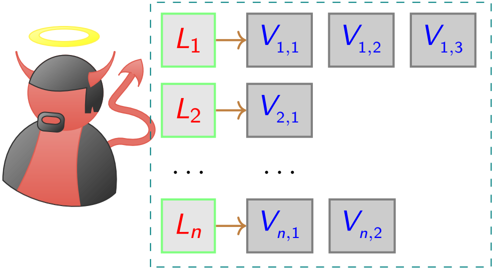
Cloud Manager

Plaintext Multi-Maps with Honest-but-Curious Cloud Manager



Plaintext Multi-Maps with Honest-but-Curious Cloud Manager



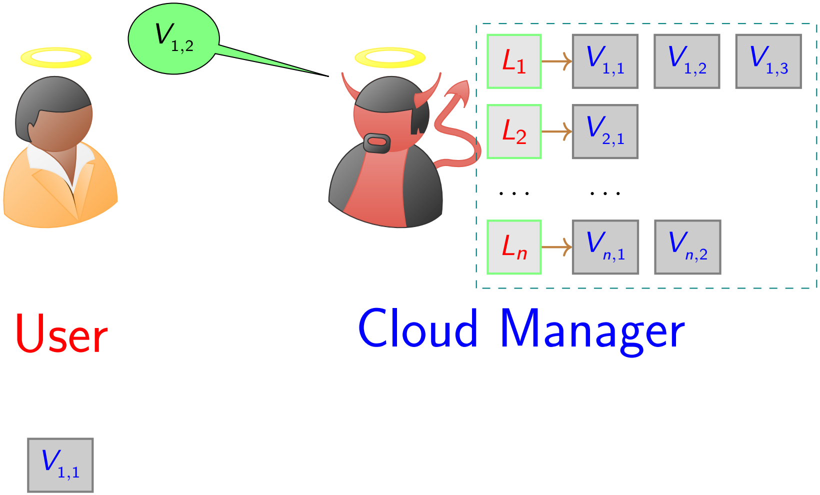


User

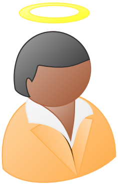
Cloud Manager

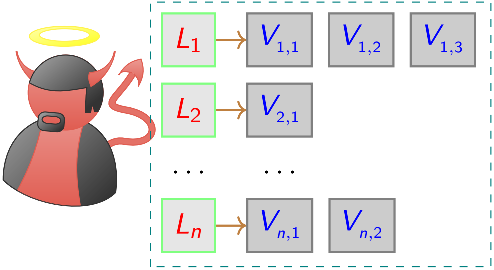
$V_{1,1}$

Plaintext Multi-Maps with Honest-but-Curious Cloud Manager



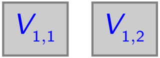
Plaintext Multi-Maps with Honest-but-Curious Cloud Manager



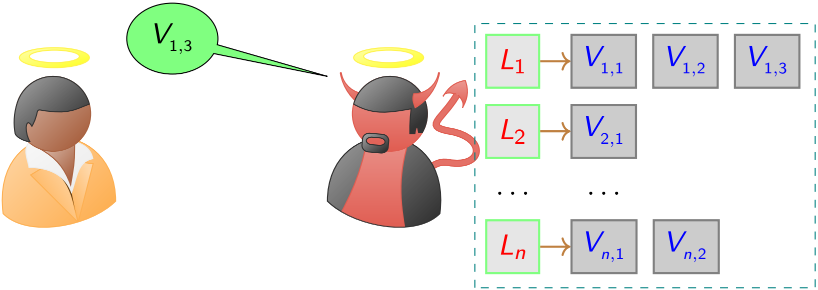


User

Cloud Manager



Plaintext Multi-Maps with Honest-but-Curious Cloud Manager

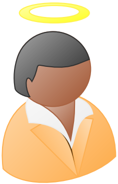


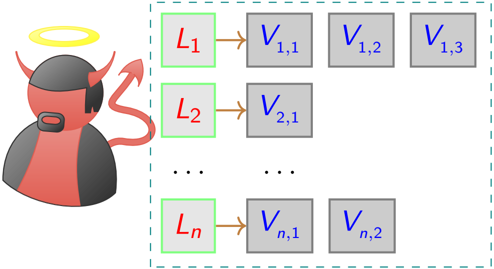
User

Cloud Manager

$V_{1,1}$ $V_{1,2}$

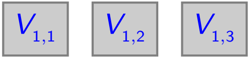
Plaintext Multi-Maps with Honest-but-Curious Cloud Manager



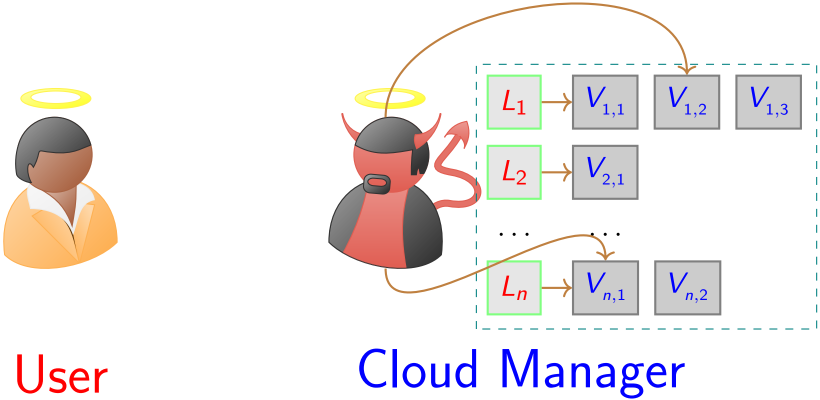


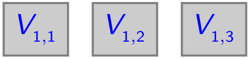
User

Cloud Manager

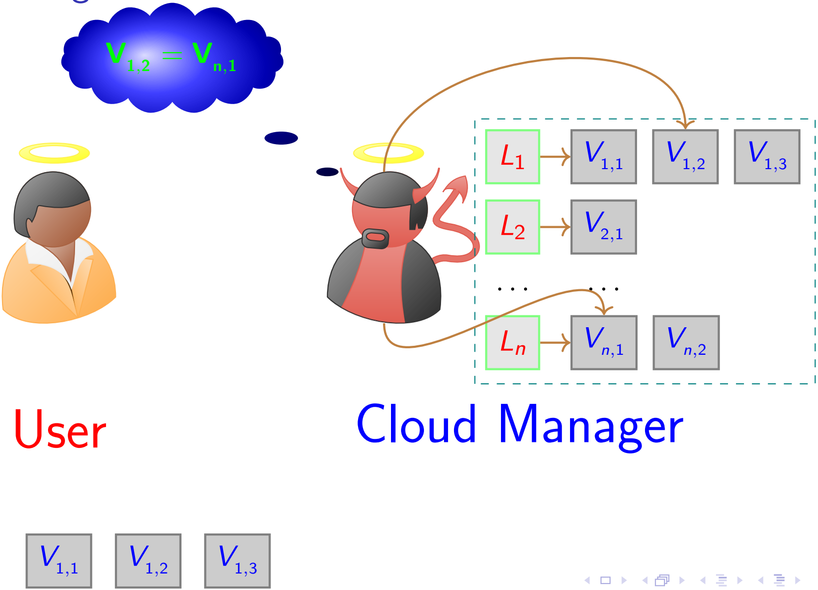


Plaintext Multi-Maps with Honest-but-Curious Cloud Manager

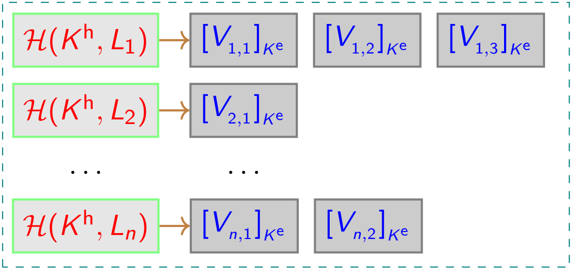




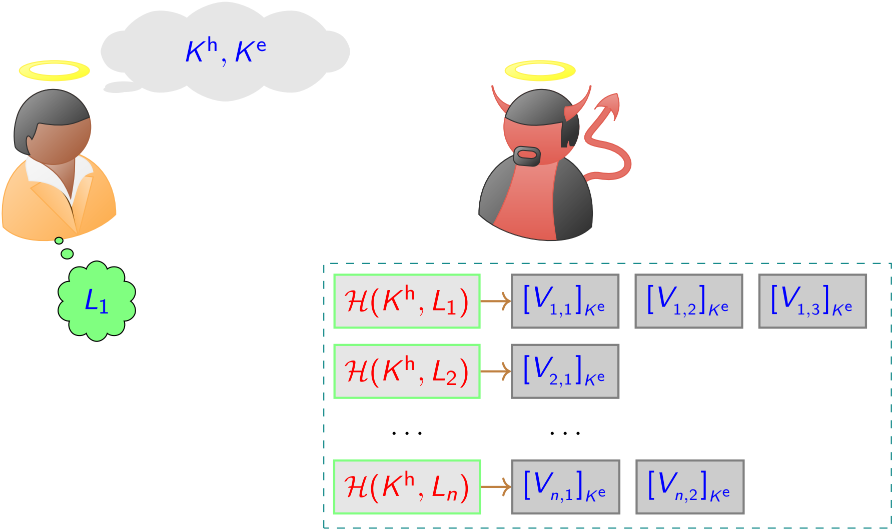
Plaintext Multi-Maps with Honest-but-Curious Cloud Manager



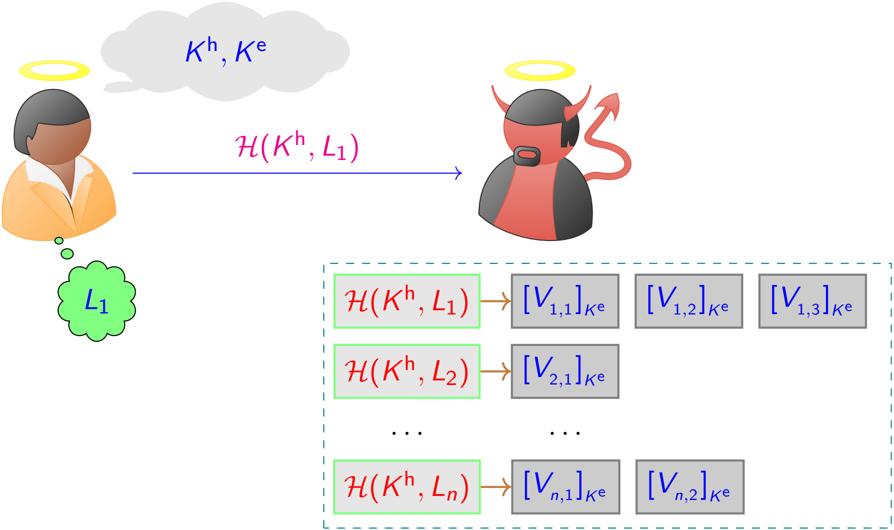
The "Hash-and-Encrypt Approach"



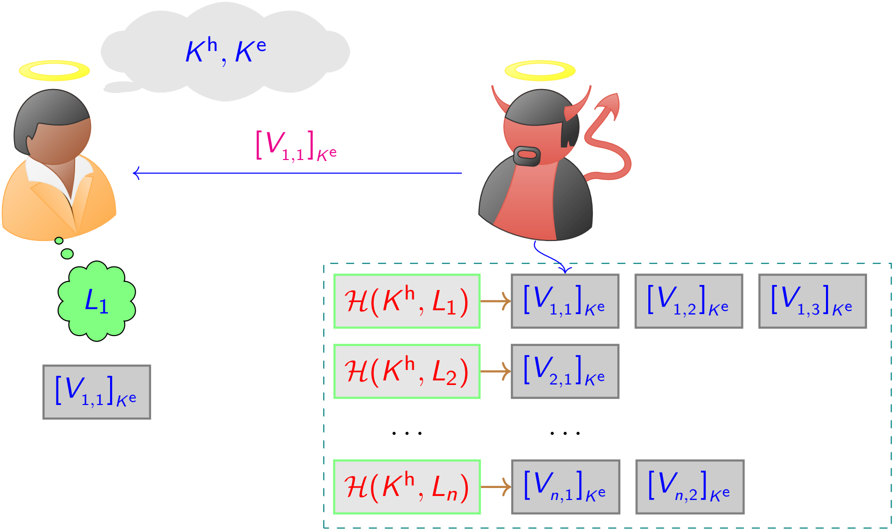
The “Hash-and-Encrypt Approach”



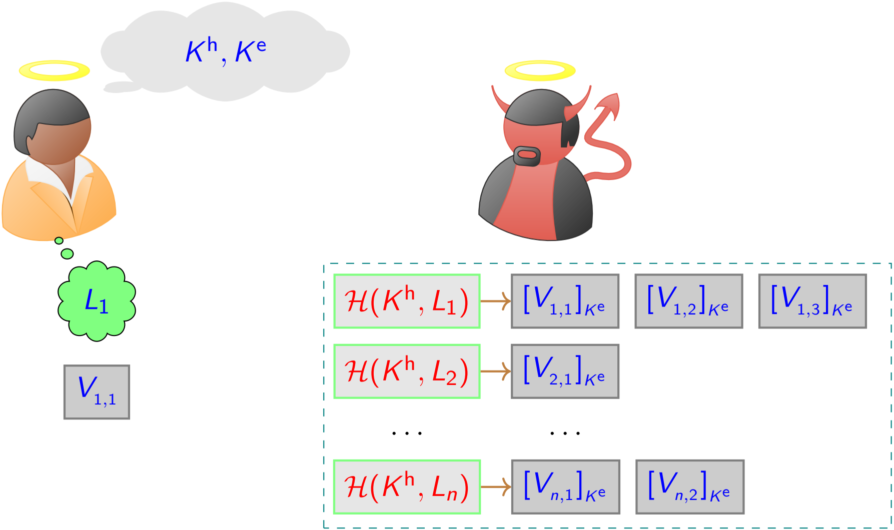
The "Hash-and-Encrypt Approach"



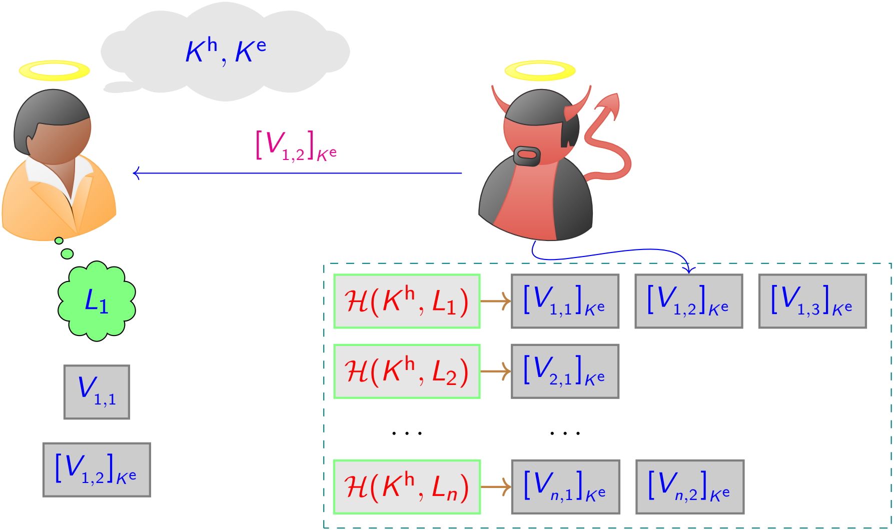
The "Hash-and-Encrypt Approach"



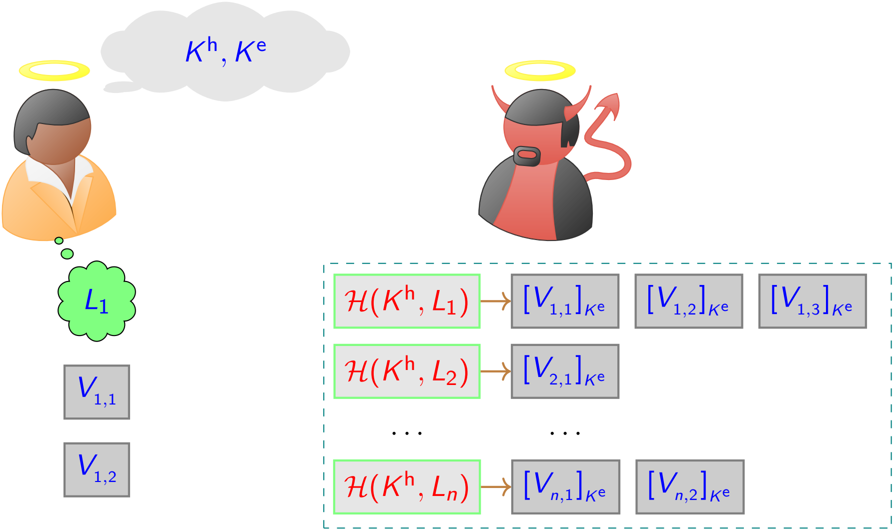
The “Hash-and-Encrypt Approach”



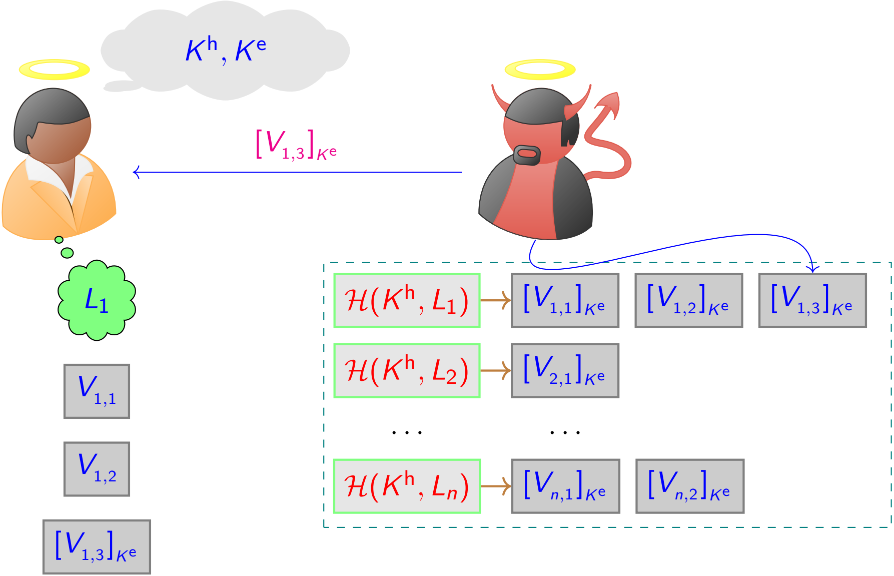
The "Hash-and-Encrypt Approach"



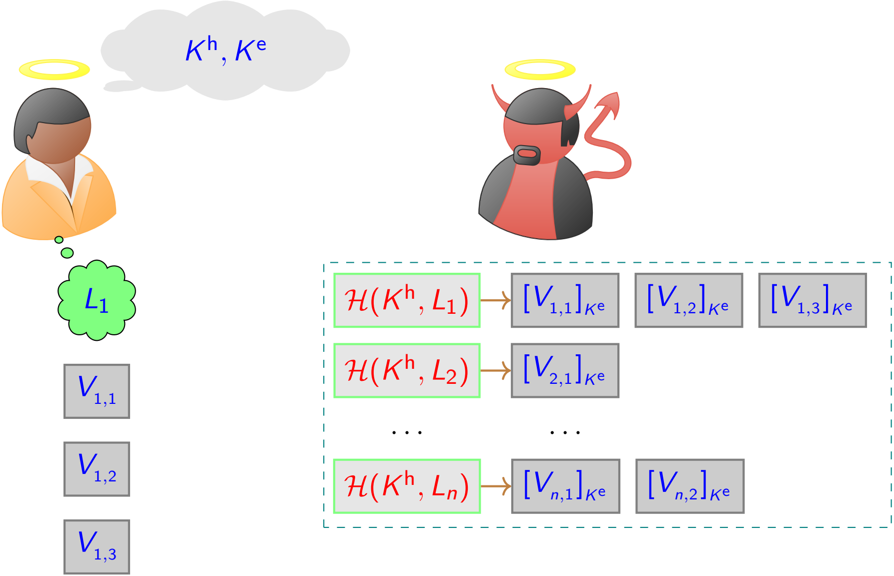
The "Hash-and-Encrypt Approach"



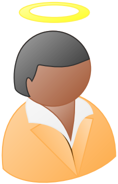
The "Hash-and-Encrypt Approach"

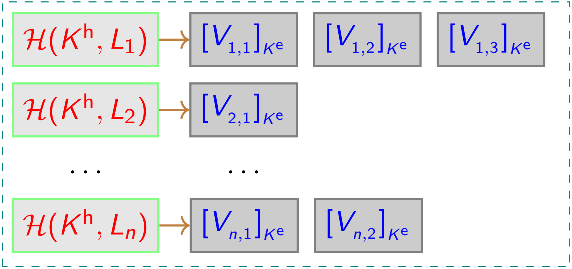


The “Hash-and-Encrypt Approach”



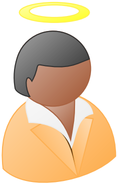
The Leakage: what the Cloud Manager learns





The Leakage: what the Cloud Manager learns

N, number of ciphertexts



$\mathcal{H}(K^{\mathrm{h}}, L_1) \rightarrow [V_{1,1}]_{K^{\mathrm{e}}}$ $[V_{1,2}]_{K^{\mathrm{e}}}$ $[V_{1,3}]_{K^{\mathrm{e}}}$

$\mathcal{H}(K^{\mathrm{h}}, L_2) \rightarrow [V_{2,1}]_{K^{\mathrm{e}}}$

$\dots$ $\dots$

$\mathcal{H}(K^{\mathrm{h}}, L_n) \rightarrow [V_{n,1}]_{K^{\mathrm{e}}}$ $[V_{n,2}]_{K^{\mathrm{e}}}$

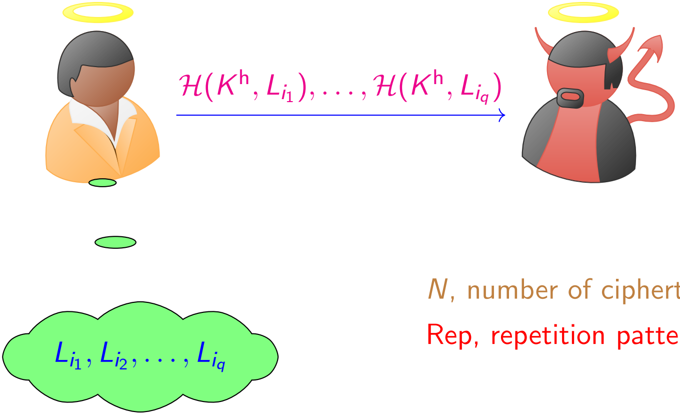
The Leakage: what the Cloud Manager learns

$\mathcal{H}(K^{\mathsf{h}}, L_{i_1}), \ldots, \mathcal{H}(K^{\mathsf{h}}, L_{i_q})$ →

$L_{i_1}, L_{i_2}, \ldots, L_{i_q}$

N, number of ciphertexts

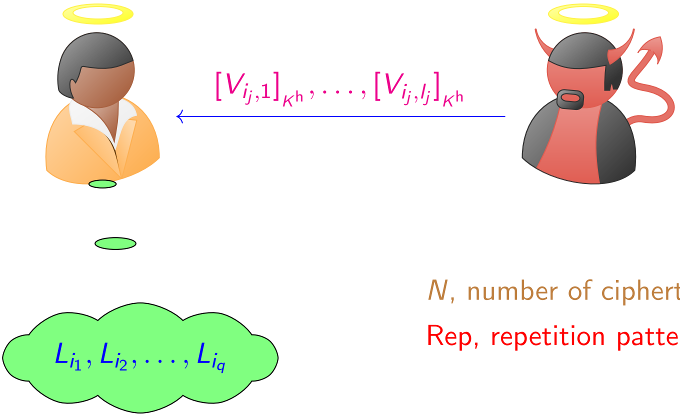
The Leakage: what the Cloud Manager learns



N, number of ciphertexts

Rep, repetition pattern

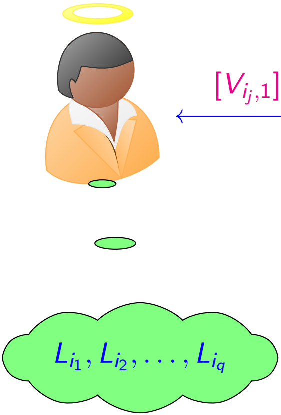
The Leakage: what the Cloud Manager learns



N, number of ciphertexts

Rep, repetition pattern

The Leakage: what the Cloud Manager learns

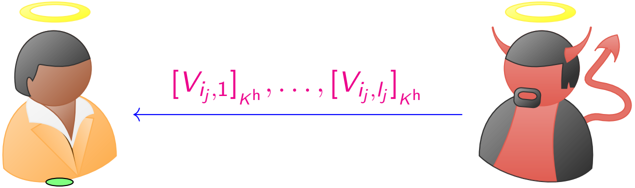


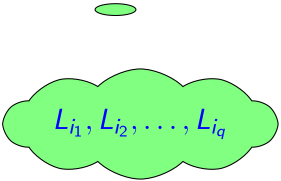
$[V_{i_j,1}]_{K^{\mathsf{h}}}, \ldots, [V_{i_j,l_j}]_{K^{\mathsf{h}}}$

N, number of ciphertexts

Rep, repetition pattern

The Leakage: what the Cloud Manager learns





N, number of ciphertexts

Rep, repetition pattern

l_j, volume of the j-th reply

The Leakage: what the Cloud Manager learns

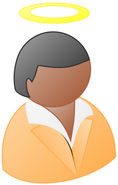
$L_{i_1}, L_{i_2}, \ldots, L_{i_q}$

N, number of ciphertexts

Rep, repetition pattern

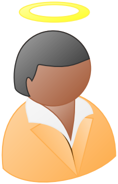
l_j, volume of the j-th reply

Padding to maximum length



$\mathcal{H}(K^{\mathsf{h}}, L_1)$ →	$[V_{1,1}]_{K^{\mathsf{e}}}$	$[V_{1,2}]_{K^{\mathsf{e}}}$	$[V_{1,3}]_{K^{\mathsf{e}}}$
$\mathcal{H}(K^{\mathsf{h}}, L_2)$ →	$[V_{2,1}]_{K^{\mathsf{e}}}$	$[0]_{K^{\mathsf{e}}}$	$[0]_{K^{\mathsf{e}}}$
...	...		
$\mathcal{H}(K^{\mathsf{h}}, L_n)$ →	$[V_{n,1}]_{K^{\mathsf{e}}}$	$[V_{n,2}]_{K^{\mathsf{e}}}$	$[0]_{K^{\mathsf{e}}}$

Padding to maximum length



$\mathcal{H}(K^{h}, L_1) \rightarrow [V_{1,1}]_{K^{e}}$ $[V_{1,2}]_{K^{e}}$ $[V_{1,3}]_{K^{e}}$

$\mathcal{H}(K^{h}, L_2) \rightarrow [V_{2,1}]_{K^{e}}$ $[0]_{K^{e}}$ $[0]_{K^{e}}$

$\dots$ $\dots$

$\mathcal{H}(K^{h}, L_n) \rightarrow [V_{n,1}]_{K^{e}}$ $[V_{n,2}]_{K^{e}}$ $[0]_{K^{e}}$

Padding to maximum length

Leakage

n

$\ell_{\max}$

Rep

$\mathcal{H}(K^{h}, L_1)$ →	$[V_{1,1}]_{K^e}$	$[V_{1,2}]_{K^e}$	$[V_{1,3}]_{K^e}$
$\mathcal{H}(K^{h}, L_2)$ →	$[V_{2,1}]_{K^e}$	$[0]_{K^e}$	$[0]_{K^e}$
$\ldots$	$\ldots$		
$\mathcal{H}(K^{h}, L_n)$ →	$[V_{n,1}]_{K^e}$	$[V_{n,2}]_{K^e}$	$[0]_{K^e}$

The Simulator

$$\mathcal{I} = (\mathcal{Q}, \text{Data})$$

The Simulator

$\mathcal{I} = (\mathcal{Q}, \text{Data})$

$L_{i_1}, \ldots, L_{i_q}$

The Simulator

$\mathcal{I} = (\mathcal{Q}, \text{Data})$

$(L_1, (V_{1,1}, \ldots, V_{1,l_1}), \ldots, (L_n, (V_{n,1}, \ldots, V_{n,l_n})))$

The Simulator

$\mathcal{I} = (\mathcal{Q}, \text{Data})$

$L_{i_1}, \ldots, L_{i_q}$

$\text{Rep} = (1, 2, 1, 2, 5, 6, 1, 8, 6, \ldots,)$

$\mathcal{H}(K^{\text{h}}, L_{i_1}), \ldots, \mathcal{H}(K^{\text{h}}, L_{i_q}))$

The Simulator

$\mathcal{I} = (\mathcal{Q}, \text{Data})$

$(L_1, (V_{1,1}, \ldots, V_{1,l_1}), \ldots, (L_n, (V_{n,1}, \ldots, V_{n,l_n})))$

$n, \ell_{\max}$

$\mathcal{H}(K^{\text{h}}, L_i), ([V_{i,1}]_{K^{\text{e}}}, \ldots, [V_{i,\ell_{\max}}]_{K^{\text{e}}})$

$\text{Rep} = (1, 2, 1, 2, 5, 6, 1, 8, 6, \ldots,)$

$\mathcal{H}(K^{\text{h}}, L_{i_1}), \ldots, \mathcal{H}(K^{\text{h}}, L_{i_q}))$

The Simulator

$\mathcal{I} = (\mathcal{Q}, \mathsf{Data})$

Leakage

$n, \ell_{\max}$

$\mathsf{Rep} = (1, 2, 1, 2, 5, 6, 1, 8, 6, \ldots,)$

View

$\mathcal{H}(K^{\mathsf{h}}, L_i), ([V_{i,1}]_{K^{\mathsf{e}}}, \ldots, [V_{i,\ell_{\max}}]_{K^{\mathsf{e}}})$

$\mathcal{H}(K^{\mathsf{h}}, L_{i_1}), \ldots, \mathcal{H}(K^{\mathsf{h}}, L_{i_q}))$

The Simulator

$\mathcal{I} = (\mathcal{Q}, \text{Data})$

Leakage

$n, \ell_{\max}$

$\text{Rep} = (1, 2, 1, 2, 5, 6, 1, 8, 6, \ldots,)$

View

$\mathcal{H}(K^{\text{h}}, L_i), ([V_{i,1}]_{K^{\text{e}}}, \ldots, [V_{i,\ell_{\max}}]_{K^{\text{e}}})$

$\mathcal{H}(K^{\text{h}}, L_{i_1}), \ldots, \mathcal{H}(K^{\text{h}}, L_{i_q}))$

Simulated View

$R_i, ([0]_{K^{\text{e}}}, \ldots, [0]_{K^{\text{e}}})$

$R_{i_1}, \ldots, R_{i_q}$

The Simulator

$\mathcal{I} = (\mathcal{Q}, \text{Data})$

View

$\mathcal{H}(K^{\text{h}}, L_i), ([V_{i,1}]_{K^{\text{e}}}, \ldots, [V_{i,\ell_{\max}}]_{K^{\text{e}}})$

$\mathcal{H}(K^{\text{h}}, L_{i_1}), \ldots, \mathcal{H}(K^{\text{h}}, L_{i_q}))$

Leakage

$n, \ell_{\max}$

$\text{Rep} = (1, 2, 1, 2, 5, 6, 1, 8, 6, \ldots,)$

Indistinguishable

Simulated View

$R_i, ([0]_{K^{\text{e}}}, \ldots, [0]_{K^{\text{e}}})$

$R_{i_1}, \ldots, R_{i_q}$

It seems we are done

Implementation of Encrypted Multi-Map

1. That leaks
   - size of the multi-map
   - query repetition pattern
2. it is volume hiding
3. security under existence of one-way functions

It seems we are done

Implementation of Encrypted Multi-Map

1. That leaks
   - size of the multi-map
   - query repetition pattern
2. it is volume hiding
3. security under existence of one-way functions

Query time is $\Theta(\ell_{\max})$

Very good!!!

It seems we are done

Implementation of Encrypted Multi-Map

1. That leaks
   - size of the multi-map
   - query repetition pattern
2. it is volume hiding
3. security under existence of one-way functions

Query time is $\Theta(\ell_{\mathrm{max}})$ Very good!!!

Storage is $\Theta(n \cdot \ell_{\mathrm{max}})$

It seems we are done

Implementation of Encrypted Multi-Map

1. That leaks
   - size of the multi-map
   - query repetition pattern
2. it is volume hiding
3. security under existence of one-way functions

Query time is $\Theta(\ell_{\mathsf{max}})$ Very good!!!

Storage is $\Theta(n \cdot \ell_{\mathsf{max}})$ Very bad!!!

Densest Subgraph Transform

[Kamara-Moataz '19]

DST

1. We have n bins
2. For each key L_i assign the $\ell_{\max}$ elements to (pseudo)-randomly chosen bins
3. Pad all bins to maximum size $\Theta(\log n)$
4. To retrieve the values for label L_i retrieve the L bins

Query time: $\Theta(L \cdot \log n)$

Concentrated Multi-Maps [K-M '19]

- (μ, τ)-Multi Maps has a set of μ keys that share τ values

Concentrated Multi-Maps [K-M '19]

- (μ, τ)-Multi Maps has a set of μ keys that share τ values
- Storage is saved by not repeating shared values

Concentrated Multi-Maps [K-M '19]

- (μ, τ)-Multi Maps has a set of μ keys that share τ values
- Storage is saved by not repeating shared values
- If values are distributed according to Zipf's law, then except with negligible probability storage is $\Theta(n)$

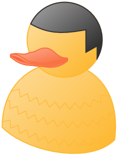
Concentrated Multi-Maps [K-M '19]

- (μ, τ)-Multi Maps has a set of μ keys that share τ values
- Storage is saved by not repeating shared values
- If values are distributed according to Zipf's law, then except with negligible probability storage is $\Theta(n)$
- Security based on hardness of planted densest subgraph

Still unhappy...

Desiderata

1. $\Theta(n)$ server storage in the worst case
2. $\Theta(\ell_{max})$ communication in the worst case
3. Standard complexity assumptions



Blueprint for Volume Hiding Multi-Maps

Dream Data Structure

- for each label L and integer j,
  there exists a set $\mathsf{Mem}(L, j)$ of constant number of memory locations where j-th value of $\mathsf{Vals}(L)$ can be found;

Blueprint for Volume Hiding Multi-Maps

Dream Data Structure

- for each label L and integer j,
  there exists a set $\mathsf{Mem}(L, j)$ of constant number of memory locations where j-th value of $\mathsf{Vals}(L)$ can be found;
- the location in $\mathsf{Mem}(L, j)$ are pseudorandom

Blueprint for Volume Hiding Multi-Maps

Dream Data Structure

- for each label L and integer j,
  there exists a set $\mathsf{Mem}(L, j)$ of constant number of memory locations where j-th value of $\mathsf{Vals}(L)$ can be found;
- the location in $\mathsf{Mem}(L, j)$ are pseudorandom
- given N items, almost all can be stored using $\Theta(N)$ memory

Encrypted Multi-Maps in Dreamland

1. **Init** for $\mathsf{Data} = ((L_1, \mathsf{Vals}(L_1)), \ldots, (L_n, \mathsf{Vals}(L_n)))$

Encrypted Multi-Maps in Dreamland

1. **Init** for $\mathsf{Data} = ((L_1, \mathsf{Vals}(L_1)), \ldots, (L_n, \mathsf{Vals}(L_n)))$
   - randomly select encryption key K^{e}

Encrypted Multi-Maps in Dreamland

1. **Init** for $\text{Data} = ((L_1, \text{Vals}(L_1)), \ldots, (L_n, \text{Vals}(L_n)))$
   - randomly select encryption key K^{e}
   - for $i = 1, \ldots, n$

     for $j = 1, \ldots, \ell_i$

     store $[L_i, V_{i,j}]_{K^{\text{e}}}$ in one of the locations of $\text{Mem}(L_i, j)$

Encrypted Multi-Maps in Dreamland

1. **Init** for Data $= ((L_1, \mathsf{Vals}(L_1)), \ldots, (L_n, \mathsf{Vals}(L_n)))$
   - randomly select encryption key K^{e}
   - for $i = 1, \ldots, n$
     for $j = 1, \ldots, \ell_i$
     store $[L_i, V_{i,j}]_{K^{\mathsf{e}}}$ in one of the locations of $\mathsf{Mem}(L_i, j)$
   - keep the few that did not fit in local stash

Encrypted Multi-Maps in Dreamland

1. **Init** for $\mathsf{Data} = ((L_1, \mathsf{Vals}(L_1)), \ldots, (L_n, \mathsf{Vals}(L_n)))$
   - randomly select encryption key K^{e}
   - for $i = 1, \ldots, n$

     for $j = 1, \ldots, \ell_i$

     store $[L_i, V_{i,j}]_{K^{\mathsf{e}}}$ in one of the locations of $\mathsf{Mem}(L_i, j)$
   - keep the few that did not fit in local stash
2. **Retrieve values for label L**

Encrypted Multi-Maps in Dreamland

1. **Init** for Data $= ((L_1, \mathsf{Vals}(L_1)), \ldots, (L_n, \mathsf{Vals}(L_n)))$
   - randomly select encryption key K^{e}
   - for $i = 1, \ldots, n$
     for $j = 1, \ldots, \ell_i$
     store $[L_i, V_{i,j}]_{K^{\mathsf{e}}}$ in one of the locations of $\mathsf{Mem}(L_i, j)$
   - keep the few that did not fit in local stash
2. **Retrieve values for label L**
   - for $j = 1, \ldots, \ell_{\max}$:

Encrypted Multi-Maps in Dreamland

1. **Init** for Data $= ((L_1, \mathsf{Vals}(L_1)), \ldots, (L_n, \mathsf{Vals}(L_n)))$
   - randomly select encryption key K^{e}
   - for $i = 1, \ldots, n$

     for $j = 1, \ldots, \ell_i$

     store $[L_i, V_{i,j}]_{K^{\mathrm{e}}}$ in one of the locations of $\mathsf{Mem}(L_i, j)$
   - keep the few that did not fit in local stash
2. **Retrieve values for label L**
   - for $j = 1, \ldots, \ell_{\max}$:
     - ask for all memory cells in $\mathsf{Mem}(L, j)$,

Encrypted Multi-Maps in Dreamland

1. **Init** for Data $= ((L_1, \mathsf{Vals}(L_1)), \ldots, (L_n, \mathsf{Vals}(L_n)))$
   - randomly select encryption key K^{e}
   - for $i = 1, \ldots, n$

     for $j = 1, \ldots, \ell_i$

     store $[L_i, V_{i,j}]_{K^{\mathsf{e}}}$ in one of the locations of $\mathsf{Mem}(L_i, j)$
   - keep the few that did not fit in local stash
2. **Retrieve values for label L**
   - for $j = 1, \ldots, \ell_{\mathsf{max}}$:
     - ask for all memory cells in $\mathsf{Mem}(L, j)$,
     - decrypt all ciphertexts received

Encrypted Multi-Maps in Dreamland

1. **Init** for Data $= ((L_1, \mathsf{Vals}(L_1)), \ldots, (L_n, \mathsf{Vals}(L_n)))$
   - randomly select encryption key K^{e}
   - for $i = 1, \ldots, n$

     for $j = 1, \ldots, \ell_i$

     store $[L_i, V_{i,j}]_{K^{\mathsf{e}}}$ in one of the locations of $\mathsf{Mem}(L_i, j)$
   - keep the few that did not fit in local stash
2. **Retrieve values for label L**
   - for $j = 1, \ldots, \ell_{\max}$:
     - ask for all memory cells in $\mathsf{Mem}(L, j)$,
     - decrypt all ciphertexts received
     - keep the ones $(L, \star)$

Encrypted Multi-Maps in Dreamland

1. **Init** for $\mathsf{Data} = ((L_1, \mathsf{Vals}(L_1)), \ldots, (L_n, \mathsf{Vals}(L_n)))$
   - randomly select encryption key K^e
   - for $i = 1, \ldots, n$
     for $j = 1, \ldots, \ell_i$
     store $[L_i, V_{i,j}]_{K^e}$ in one of the locations of $\mathsf{Mem}(L_i, j)$
   - keep the few that did not fit in local stash
2. **Retrieve values for label L**
   - for $j = 1, \ldots, \ell_{\max}$:
     - ask for all memory cells in $\mathsf{Mem}(L, j)$,
     - decrypt all ciphertexts received
     - keep the ones $(L, \star)$
     - look for the missing ones in the stash

Encrypted Multi-Maps in Dreamland

1. **Init** for Data $= ((L_1, \text{Vals}(L_1)), \ldots, (L_n, \text{Vals}(L_n)))$
   - randomly select encryption key K^e
   - for $i = 1, \ldots, n$
     for $j = 1, \ldots, \ell_i$
     store $[L_i, V_{i,j}]_{K^e}$ in one of the locations of $\text{Mem}(L_i, j)$
   - keep the few that did not fit in local stash
2. **Retrieve values for label L**
   - for $j = 1, \ldots, \ell_{\text{max}}$:
     - ask for all memory cells in $\text{Mem}(L, j)$,
     - decrypt all ciphertexts received
     - keep the ones $(L, \star)$
     - look for the missing ones in the stash

Encrypted Multi-Maps in Dreamland

1. **Init** for Data $= ((L_1, \text{Vals}(L_1)), \ldots, (L_n, \text{Vals}(L_n)))$
   - randomly select encryption key K^e
   - for $i = 1, \ldots, n$

     for $j = 1, \ldots, \ell_i$

     store $[L_i, V_{i,j}]_{K^e}$ in one of the locations of $\text{Mem}(L_i, j)$
   - keep the few that did not fit in local stash
2. **Retrieve values for label L**
   - for $j = 1, \ldots, \ell_{\max}$:
     - ask for all memory cells in $\text{Mem}(L, j)$,
     - decrypt all ciphertexts received
     - keep the ones $(L, \star)$
     - look for the missing ones in the stash

Server memory: $\Theta(N)$, $N = \ell_1 + \ldots, \ell_n$

Encrypted Multi-Maps in Dreamland

1. **Init** for Data $= ((L_1, \mathsf{Vals}(L_1)), \ldots, (L_n, \mathsf{Vals}(L_n)))$
   - randomly select encryption key K^e
   - for $i = 1, \ldots, n$
     for $j = 1, \ldots, \ell_i$
     store $[L_i, V_{i,j}]_{K^e}$ in one of the locations of $\mathsf{Mem}(L_i, j)$
   - keep the few that did not fit in local stash
2. **Retrieve values for label L**
   - for $j = 1, \ldots, \ell_{\max}$:
     - ask for all memory cells in $\mathsf{Mem}(L, j)$,
     - decrypt all ciphertexts received
     - keep the ones $(L, \star)$
     - look for the missing ones in the stash

Server memory: $\Theta(N)$, $N = \ell_1 + \ldots, \ell_n$
Query bandwidth: $\Theta(\ell_{\max})$

Encrypted Multi-Maps in Dreamland

1. **Init** for Data $= ((L_1, \mathsf{Vals}(L_1)), \ldots, (L_n, \mathsf{Vals}(L_n)))$
   - randomly select encryption key K^{e}
   - for $i = 1, \ldots, n$

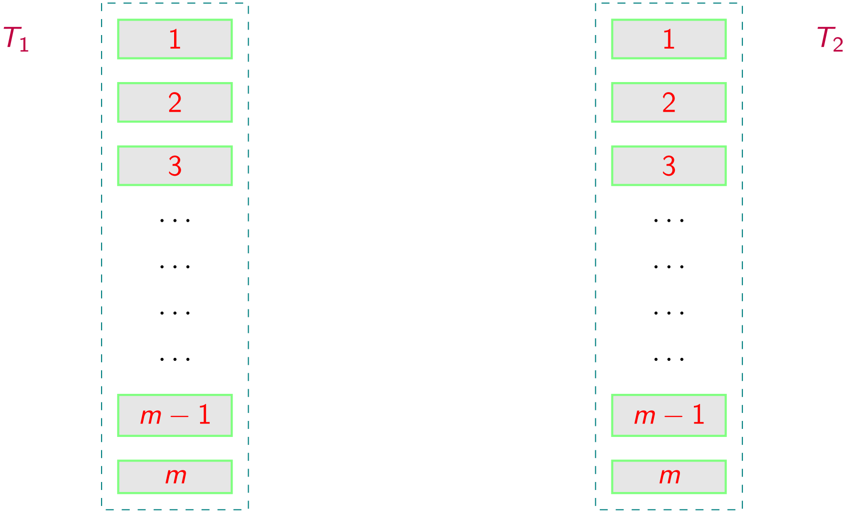
     for $j = 1, \ldots, \ell_i$

     store $[L_i, V_{i,j}]_{K^{\mathsf{e}}}$ in one of the locations of $\mathsf{Mem}(L_i, j)$
   - keep the few that did not fit in local stash
2. **Retrieve values for label L**
   - for $j = 1, \ldots, \ell_{\max}$:
     - ask for all memory cells in $\mathsf{Mem}(L, j)$,
     - decrypt all ciphertexts received
     - keep the ones $(L, \star)$
     - look for the missing ones in the stash

Server memory: $\Theta(N)$, $N = \ell_1 + \ldots, \ell_n$
Query bandwidth: $\Theta(\ell_{\max})$
Client memory: few values

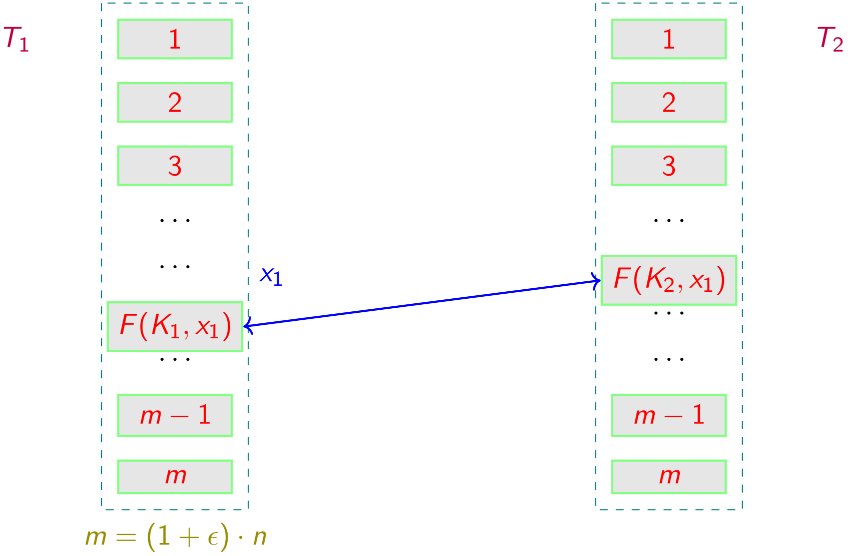
Enter Cuckoo Hashing

The Cuckoo Graph for n items

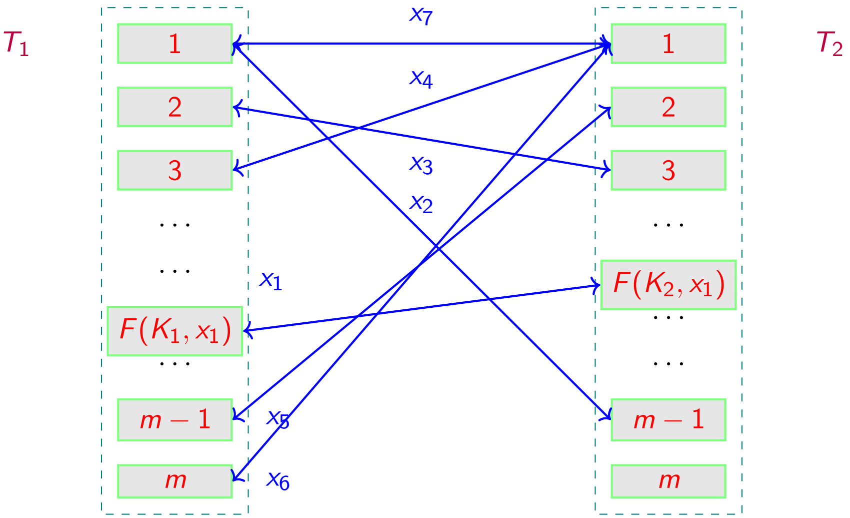


$m = (1+\epsilon) \cdot n$

The Cuckoo Graph for n items



The Cuckoo Graph for n items



$m = (1+\epsilon) \cdot n$

Cuckoo Graph

- take each component of the cuckoo graph

Cuckoo Graph

- take each component of the cuckoo graph
  - edges correspond to items

Cuckoo Graph

- take each component of the cuckoo graph
  - edges correspond to items
  - vertices correspond to memory slots

Cuckoo Graph

- take each component of the cuckoo graph
  - edges correspond to items
  - vertices correspond to memory slots
- if it is a tree or if it has only one cycle

Cuckoo Graph

- take each component of the cuckoo graph
  - edges correspond to items
  - vertices correspond to memory slots
- if it is a tree or if it has only one cycle
  - number of edges $\leq$ number of vertices

Cuckoo Graph

- take each component of the cuckoo graph
  - edges correspond to items
  - vertices correspond to memory slots
- if it is a tree or if it has only one cycle
  - number of edges $\leq$ number of vertices
  - there is enough space to store each item in one of the two endpoints

Cuckoo Graph

- take each component of the cuckoo graph
  - edges correspond to items
  - vertices correspond to memory slots
- if it is a tree or if it has only one cycle
  - number of edges $\leq$ number of vertices
  - there is enough space to store each item in one of the two endpoints
- if it is has more than only one cycle

Cuckoo Graph

- take each component of the cuckoo graph
  - edges correspond to items
  - vertices correspond to memory slots
- if it is a tree or if it has only one cycle
  - number of edges $\leq$ number of vertices
  - there is enough space to store each item in one of the two endpoints
- if it is has more than only one cycle
  - remove edges until we fall in the previous case

Cuckoo Graph

- take each component of the cuckoo graph
  - edges correspond to items
  - vertices correspond to memory slots
- if it is a tree or if it has only one cycle
  - number of edges $\leq$ number of vertices
  - there is enough space to store each item in one of the two endpoints
- if it is has more than only one cycle
  - remove edges until we fall in the previous case
  - removed items are stored in the stash

Cuckoo Graph

- take each component of the cuckoo graph
  - edges correspond to items
  - vertices correspond to memory slots
- if it is a tree or if it has only one cycle
  - number of edges $\leq$ number of vertices
  - there is enough space to store each item in one of the two endpoints
- if it is has more than only one cycle
  - remove edges until we fall in the previous case
  - removed items are stored in the stash

Cuckoo Graph

- take each component of the cuckoo graph
  - edges correspond to items
  - vertices correspond to memory slots
- if it is a tree or if it has only one cycle
  - number of edges $\leq$ number of vertices
  - there is enough space to store each item in one of the two endpoints
- if it is has more than only one cycle
  - remove edges until we fall in the previous case
  - removed items are stored in the stash

Theorem (Kirsch-Mitzenmacher-Wieder '09)

The probability that s items are stored in the stash is $O(n^{-s})$.

Constructing the Cuckoo Hash Table

- The construction of the components of the cuckoo graph and the deletion of the extra edges can be performed with a sequence of MapReduce operations

Constructing the Cuckoo Hash Table

- The construction of the components of the cuckoo graph and the deletion of the extra edges can be performed with a sequence of MapReduce operations
- MapReduce can be performed obliviously

Constructing the Cuckoo Hash Table

- The construction of the components of the cuckoo graph and the deletion of the extra edges can be performed with a sequence of MapReduce operations
- MapReduce can be performed obliviously
- In practice (and in our experiments):

Constructing the Cuckoo Hash Table

- The construction of the components of the cuckoo graph and the deletion of the extra edges can be performed with a sequence of MapReduce operations
- MapReduce can be performed obliviously
- In practice (and in our experiments):
  - try to insert x to X_1 or X_2

Constructing the Cuckoo Hash Table

- The construction of the components of the cuckoo graph and the deletion of the extra edges can be performed with a sequence of MapReduce operations
- MapReduce can be performed obliviously
- In practice (and in our experiments):
  - try to insert x to X_1 or X_2
  - if one is empty, we are done

Constructing the Cuckoo Hash Table

- The construction of the components of the cuckoo graph and the deletion of the extra edges can be performed with a sequence of MapReduce operations
- MapReduce can be performed obliviously
- In practice (and in our experiments):
  - try to insert x to X_1 or X_2
  - if one is empty, we are done
  - otherwise evict y from X_1

Constructing the Cuckoo Hash Table

- The construction of the components of the cuckoo graph and the deletion of the extra edges can be performed with a sequence of MapReduce operations
- MapReduce can be performed obliviously
- In practice (and in our experiments):
  - try to insert x to X_1 or X_2
  - if one is empty, we are done
  - otherwise evict y from X_1
  - try to insert y to Y_2

Constructing the Cuckoo Hash Table

- The construction of the components of the cuckoo graph and the deletion of the extra edges can be performed with a sequence of MapReduce operations
- MapReduce can be performed obliviously
- In practice (and in our experiments):
  - try to insert x to X_1 or X_2
  - if one is empty, we are done
  - otherwise evict y from X_1
  - try to insert y to Y_2
  - if not successful after M steps, add x to stash

Constructing the Cuckoo Hash Table

- The construction of the components of the cuckoo graph and the deletion of the extra edges can be performed with a sequence of MapReduce operations
- MapReduce can be performed obliviously
- In practice (and in our experiments):
  - try to insert x to X_1 or X_2
  - if one is empty, we are done
  - otherwise evict y from X_1
  - try to insert y to Y_2
  - if not successful after M steps, add x to stash
- If $M = \Omega(\log n)$ then resulting stash is very small and average insertion time stays constant.

Blueprint for Volume Hiding Multi-Maps – Revisited

Cuckoo Hashing

- for each label L and integer j,
  there exists a set $\mathsf{Mem}(L, j)$ of constant number of memory locations where j-th value of $\mathsf{Vals}(L)$ can be found;

Blueprint for Volume Hiding Multi-Maps – Revisited

Cuckoo Hashing

- for each label L and integer j,
  there exists a set $\mathsf{Mem}(L, j)$ of two memory locations where j-th value of $\mathsf{Vals}(L)$ can be found;
- the location in $\mathsf{Mem}(L, j)$ are pseudorandom

Blueprint for Volume Hiding Multi-Maps – Revisited

Cuckoo Hashing

- for each label L and integer j,
  there exists a set $\text{Mem}(L, j)$ of two memory locations where j-th value of $\text{Vals}(L)$ can be found;
- the location in $\text{Mem}(L, j)$ are pseudorandom
- given N items, almost all can be stored using $\Theta(N)$ memory

Encrypted Multi-Maps using Cuckoo Hashing

Algorithm **Init**

Data $= ((L_1, \mathsf{Vals}(L_1)), \ldots, (L_n, \mathsf{Vals}(L_n)))$

1. randomly select seeds K_1, K_2 for PRF F
2. randomly select encryption key K^{e}
3. for $i = 1, \ldots, n$
   - randomly select permutation Π_i over $[1, \ldots, \ell_{\max}]$
   - for $j = 1$ to l_i
     - Add edge labeled $[L_i, V_{i,j}]_{K^{\mathsf{e}}}$ between vertices $F(K_1, (L, \Pi_i(j)))$ and $F(K_2, (L, \Pi_i(j)))$
4. Construct T_1 and T_2 (stored remotely) and stash (stored locally)

Encrypted Multi-Maps using Cuckoo Hashing

Algorithm **Get**

Retrieve values for label L

- for $j = 1, \ldots, \ell_{\mathsf{max}}$:
  - ask for slot $F(K_1, (L, j))$ in table T_1 and slot $F(K_2, (L, j))$ in table T_2
- decrypt all ciphertexts received
- keep the ones $(L, \star)$
- look for the missing ones in the stash

Encrypted Multi-Maps using Cuckoo Hashing

1. Leakage
   - N, total number of values
   - L, maximum volume
   - Rep, query repetition pattern
2. Storage
   - Server: $(2 + \epsilon)N$
   - Client: practically constant
3. Communication
   - Client to Server $2L$ indices
   - Server to Client $2L$ ciphertexts

Security assuming One-Way Functions

Encrypted Multi-Maps using Cuckoo Hashing

1. Leakage
   - N, total number of values
   - L, maximum volume
   - Rep, query repetition pattern
2. Storage
   - Server: $(2+\epsilon)N$
   - Client: practically constant
3. Communication
   - Client to Server 2 seeds (by using delegatable PRFs)
   - Server to Client $2L$ ciphertexts

Security assuming One-Way Functions

Always download maximum volume?

Always download maximum volume?

- All previous schemes consider **perfect** volume-hiding privacy

Always download maximum volume?

- All previous schemes consider **perfect** volume-hiding privacy
- This requires that all queries download $\geq \ell_{\text{max}}$ entries

Always download maximum volume?

- All previous schemes consider **perfect** volume-hiding privacy
- This requires that all queries download $\geq \ell_{\mathsf{max}}$ entries
- This very wasteful when there is a huge variation in the length of tuples (e.g., Zipf's law)

Always download maximum volume?

- All previous schemes consider **perfect** volume-hiding privacy
- This requires that all queries download $\geq \ell_{\mathsf{max}}$ entries
- This very wasteful when there is a huge variation in the length of tuples (e.g., Zipf's law)
- **Question**:
  Can we obtain some meaningful privacy notion that allows for downloading $\leq \ell_{max}$ entries?

(ϵ, δ)-Differentially Private Volume-Hiding Encrypted Multi-Maps

Data^1 and Data^2 differ in the volume of one label L_i

$$|l_i^1 - l_i^2| = 1$$

then

$$\mathrm{Prob}[\mathsf{View}(\mathsf{Data}^1) \in S] \leq e^{\epsilon} \cdot \mathrm{Prob}[\mathsf{View}(\mathsf{Data}^2) \in S] + \delta$$

for all subsets S of views

Cuckoo hashing with DP

To retrieve tuple for label L,

$$\begin{array}{ll} F(K_1, L||1) & F(K_2, L||1) \\ F(K_1, L||2) & F(K_2, L||2) \\ F(K_1, L||3) & F(K_2, L||3) \\ \dots & \dots \\ F(K_1, L||\ell_{max}) & F(K_2, L||\ell_{max}) \end{array}$$

Cuckoo hashing with DP

To retrieve tuple for label L,

$$
\begin{array}{ll}
F(K_1, L||1) & F(K_2, L||1) \\
F(K_1, L||2) & F(K_2, L||2) \\
F(K_1, L||3) & F(K_2, L||3) \\
\dots & \dots \\
F(K_1, L||\ell_{\max} + Z_L) & F(K_2, L||\ell_{\max} + Z_L)
\end{array}
$$

Z_L is the *noise* from Laplacian$(O(1/\epsilon))$ dist.

Cuckoo hashing with DP

To retrieve tuple for label L,

$$
\begin{array}{ll}
F(K_1, L||1) & F(K_2, L||1) \\
F(K_1, L||2) & F(K_2, L||2) \\
F(K_1, L||3) & F(K_2, L||3) \\
\dots & \dots \\
F(K_1, L||\ell_{\max} + Z_L) & F(K_2, L||\ell_{\max} + Z_L)
\end{array}
$$

Z_L is the *noise* from Laplacian$(O(1/\epsilon))$ dist.

Z_L could be negative

Cuckoo hashing with DP

To retrieve tuple for label L,

$$\begin{array}{ll} F(K_1, L||1) & F(K_2, L||1) \\ F(K_1, L||2) & F(K_2, L||2) \\ F(K_1, L||3) & F(K_2, L||3) \\ \dots & \dots \\ F(K_1, L||\ell_{\max} + T + Z_L) & F(K_2, L||\ell_{\max} + T + Z_L) \end{array}$$

Z_L is the *noise* from Laplacian$(O(1/\epsilon))$ dist.

Z_L could be negative

Make sure $T \geq |Z_L|$

$|Z_L| > \log n$ with negligible probability

Cuckoo hashing with DP

Data is Sanitized

- Z_L is distributed according to Laplacian$O(1/\epsilon)$

Cuckoo hashing with DP

Data is Sanitized

- Z_L is distributed according to Laplacian $O(1/\epsilon)$
- Sampled once and stored over the server

Cuckoo hashing with DP

Data is Sanitized

- Z_L is distributed according to Laplacian $O(1/\epsilon)$
- Sampled once and stored over the server
- We need a dictionary to store it

Cuckoo hashing with DP

Data is Sanitized

- Z_L is distributed according to Laplacian$O(1/\epsilon)$
- Sampled once and stored over the server
- We need a dictionary to store it
- No volume leakage

Experiments

Experiments

Volume Hiding with dPRF vs DST

Cuckoo Hash with $m = 1.3n$

Give up insertion after $5 \log n$ tries

- **Less Server Storage**

Experiments

Volume Hiding with dPRF vs DST

Cuckoo Hash with $m = 1.3n$
Give up insertion after $5 \log n$ tries

- **Less Server Storage**
  - For $N \approx$ 4Million values
    348MB vs 384MB

Experiments

Volume Hiding with dPRF vs DST

Cuckoo Hash with $m = 1.3n$

Give up insertion after $5 \log n$ tries

- **Less Server Storage**
  - For $N \approx$ 4Million values

    348MB vs 384MB
- **Query Overhead:** 2 **ciphertexts per value (64 bytes)**

Experiments

Volume Hiding with dPRF vs DST

Cuckoo Hash with $m = 1.3n$
Give up insertion after $5 \log n$ tries

- **Less Server Storage**
  - For $N \approx$ 4Million values
    348MB vs 384MB
- **Query Overhead:** 2 **ciphertexts per value (64 bytes)**
  - 675 bytes for $N \approx$ 64000 values (10x improvement)

Experiments

Volume Hiding with dPRF vs DST

Cuckoo Hash with $m = 1.3n$

Give up insertion after $5 \log n$ tries

- **Less Server Storage**
  - For $N \approx$ 4Million values

    348MB vs 384MB
- **Query Overhead:** 2 **ciphertexts per value (64 bytes)**
  - 675 bytes for $N \approx 64000$ values (10x improvement)
  - 1008 bytes for $N \approx$ 4Million values (16x improvement)

Experiments

Volume Hiding with dPRF vs DST

Cuckoo Hash with $m = 1.3n$
Give up insertion after $5 \log n$ tries

- **Less Server Storage**
  - For $N \approx$ 4Million values
    348MB vs 384MB
- **Query Overhead:** 2 **ciphertexts per value (64 bytes)**
  - 675 bytes for $N \approx 64000$ values (10x improvement)
  - 1008 bytes for $N \approx$ 4Million values (16x improvement)
- **Client Storage**

Experiments

Volume Hiding with dPRF vs DST

Cuckoo Hash with $m = 1.3n$
Give up insertion after $5 \log n$ tries

- **Less Server Storage**
  - For $N \approx$ 4Million values
    348MB vs 384MB
- **Query Overhead:** 2 **ciphertexts per value (64 bytes)**
  - 675 bytes for $N \approx 64000$ values (10x improvement)
  - 1008 bytes for $N \approx$ 4Million values (16x improvement)
- **Client Storage**
  - less than 4 KB

Differentially-Private Volume Hiding with dPRF vs DST

- $\epsilon = 0.2$

Differentially-Private Volume Hiding with dPRF vs DST

- $\epsilon = 0.2$
- Lossy with probability 2^{-64}

Differentially-Private Volume Hiding with dPRF vs DST

- $\epsilon = 0.2$
- Lossy with probability 2^{-64}
- Number of values of a label follows Zipf's distribution

  Average length 8

Differentially-Private Volume Hiding with dPRF vs DST

- $\epsilon = 0.2$
- Lossy with probability 2^{-64}
- Number of values of a label follows Zipf's distribution
  Average length 8
- To obtain 2^{-64}, $T = 5610$,

Differentially-Private Volume Hiding with dPRF vs DST

- $\epsilon = 0.2$
- Lossy with probability 2^{-64}
- Number of values of a label follows Zipf's distribution
  Average length 8
- To obtain 2^{-64}, $T = 5610$,
  - average download is 5618

Differentially-Private Volume Hiding with dPRF vs DST

- $\epsilon = 0.2$
- Lossy with probability 2^{-64}
- Number of values of a label follows Zipf's distribution
  Average length 8
- To obtain 2^{-64}, $T = 5610$,
  - average download is 5618
- Volume-Hiding forced to download max length ≈ 84000
  15x improvement

Experiments

	Densest Subgraph Transform [KM19]				dprfMM				dpMM			
Input Multi-Map												
Number of Values (n)	2^{16}	2^{18}	2^{20}	2^{22}	2^{16}	2^{18}	2^{20}	2^{22}	2^{16}	2^{18}	2^{20}	2^{22}
Plaintext Raw Byte Size (MB)	1.05	4.19	16.78	67.11	1.05	4.19	16.78	67.11	1.05	4.19	16.78	67.11
EMM Storage												
Server (MB)	5.53	22.74	88.25	384.40	5.45	21.81	87.24	348.97	6.81	27.26	109.05	436.21
Client Stash (KB)	N/A	N/A	N/A	N/A	0.16	0.50	1.52	4.84	0.21	0.63	1.97	6.18
Query Communication												
Upload (Bytes)	16	16	16	16	16	16	16	16	36	36	36	36
Download (Bytes Per Result)	675.2	780.8	841.6	1008.0	64	64	64	64	64	64	64	64
CPU Costs												
Query (Client ms)	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
Reply (Server ms Per Result)	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
Result (Client ms Per Result)	0.01	0.01	0.01	0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01

Table 2: Microbenchmarks for server and network costs comparing volume-hiding STE schemes. We denote n as the total number of values in the input multi-map. If ℓ is the maximum volume of any key, then the first two column constructions must download ℓ results. On the other hand, the number of results for dpMM will be significantly smaller than ℓ on average. The above results apply for any input multi-map structure, query distribution as well as any value ℓ. We denote milliseconds by ms.

Thank You

ePrint: https://eprint.iacr.org/2019/1292

CCS '19: https://doi.org/10.1145/3319535.3354213