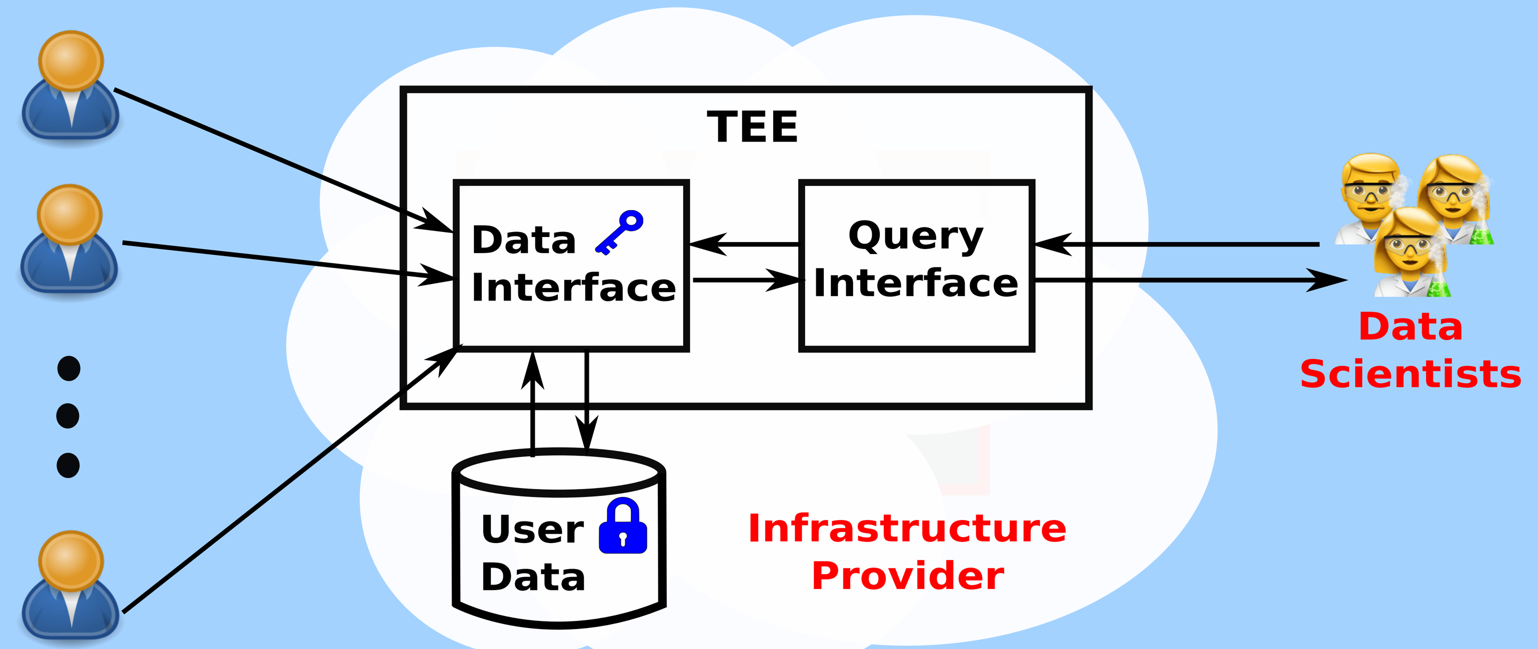
Oblivious Sampling Algorithms for Private Data Analysis

Sajin Sasy and Olga Ohrimenko

Contributions

- Introduce Private Sampling-Based Query Framework
- Take advantage of Differential Privacy and *Privacy Amplification* from sampling
- Design secure sampling for hiding sample identity: *oblivious sampling*
- Design *efficient* dataset sampling algorithms
- Experimental evaluations of accuracy of machine learning models trained with minibatches produced by *sampling instead of shuffling*

Threat Model / Framework Architecture



Goal & Motivation

- Enable data scientists to query data while providing strong privacy guarantees on user data.
- Trusted Execution Environments (TEE) restricts data access and protects data while its computed upon.
- TEEs can leak data access patterns which lead to privacy loss.

Privacy via Sampling and Differential Privacy

Consider a randomized mechanism $\mathcal{M}$ that is (ϵ,δ)-DP, and a mechanism $\mathcal{M}'$ that uses samples of size m from a dataset with n elements using any of the following methods:

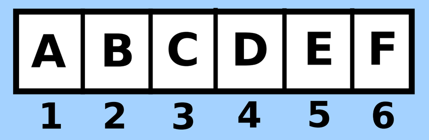
1) Sampling without Replacement (SWO)
2) Poisson Sampling
3) Shuffle-based Sampling

Informally, *Privacy Amplification of Differential Privacy* states that $\mathcal{M}'$ with SWO or Poisson sampling provides an order of $O(\sqrt{m/n})$ smaller epsilon than the popularly used Shuffle-based sampling.

Samples must be hidden for privacy amplification results to hold!

Algorithm for Oblivious Sampling Without Replacement (SWO)

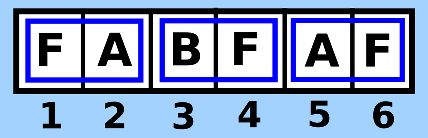
Input : A dataset (D) of n records



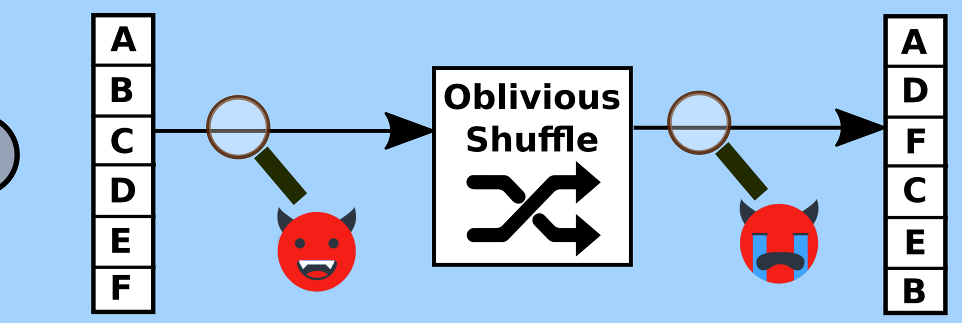
Goal: Obtain samples of size m. (m=2)

Constraint: Hide sample identity. Memory access patterns of the algorithm execution should not reveal any information about the elements in any sample.

Output : k = n/m samples of size m from D. (k=3)

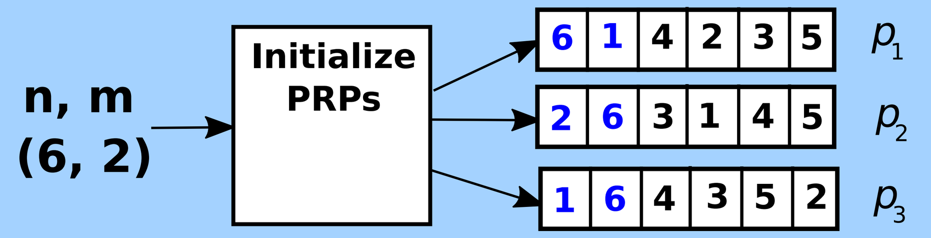


I



Shuffle the array without memory access patterns leaking any information about the output permutation

II



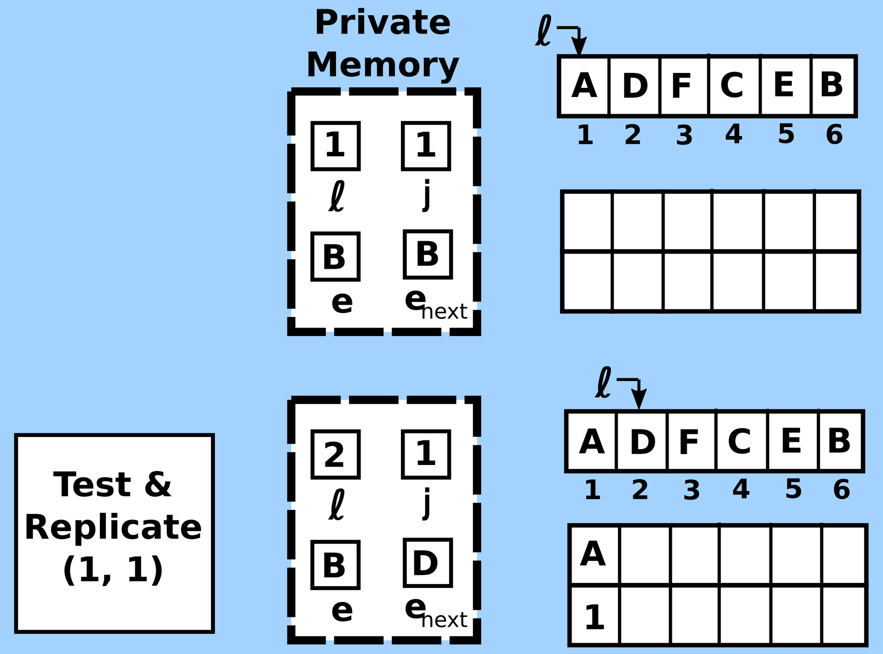
Instantiate k permutations of n using Pseudo Random Permutations (PRP), effectively contributing the indices for the sample.

III

For i = [1, k]: Test&Replicate(i,j)

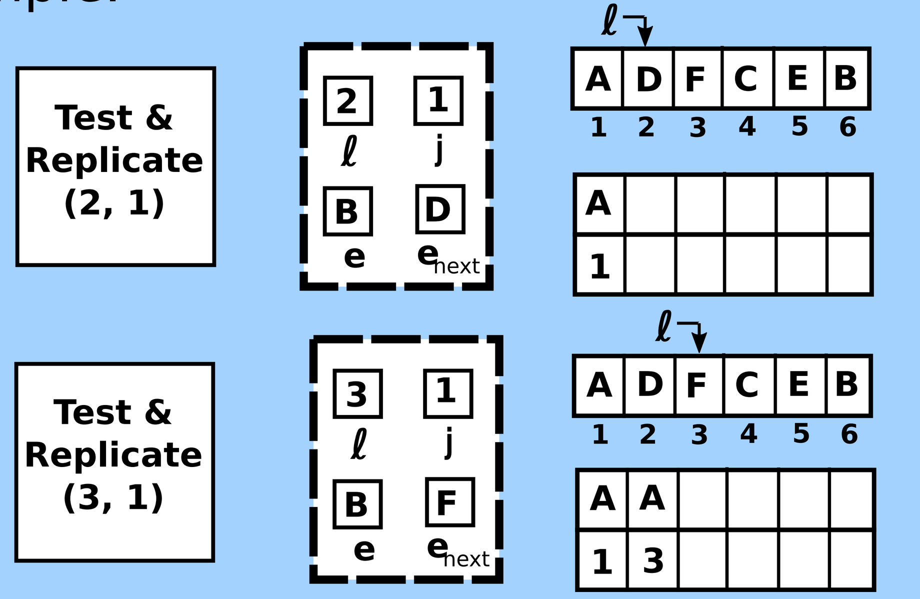
Test & Replicate (i, j)

This tool checks if sample i, contains key at index j with the PRPs generated with Initialize and replicates the key if true.

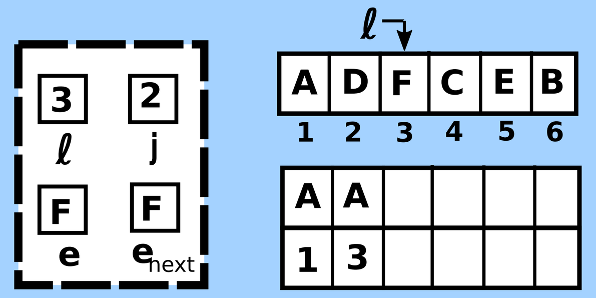


NOTE: All the elements and sample indices are encrypted by the TEE and only in clear in private memory which is not visible to the adversary.

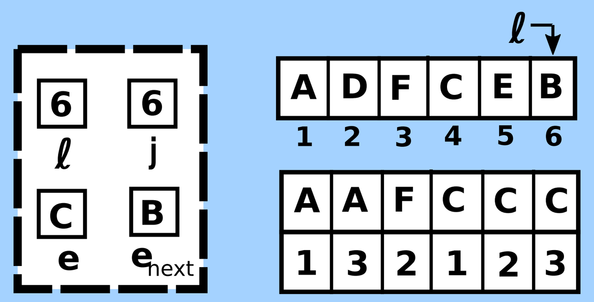
k iterations of Test & Replicate for index j produces a copy of j for every sample it appears in. Each copy is created with its sample index.



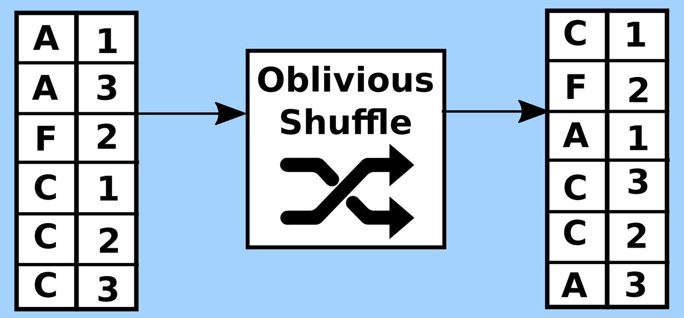
IV **Setup next element to replicate**



Repeat III and IV till ℓ = n

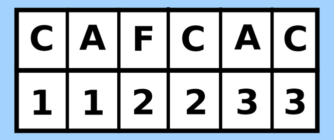


V **Obliviously shuffle the sampled array**



This shuffle breaks the correlation of which elements go to which samples.

VI **Decrypt and sort by the sample indices**



Algorithm Correctness

We prove that the algorithm returns samples of size m drawn truly randomly from the n elements, upto an injective and random key mapping.

Experimental Results

	Shuffle	SWO	Poisson
MNIST	97.50 (98.33)	97.43 (98.31)	97.47 (98.31)
DP MNIST	94.06 (94.10)	94.03 (94.05)	94.10 (94.01)
ϵ	9.39	2.13	0.82

CIFAR-10	79.6 (83.2)	79.0 (82.9)
DP CIFAR-10	73.4 (72.3)	72.5 (71)
ϵ	9.39	4.89

Full Paper

- Oblivious Poisson sampling algorithm
- Security analysis
- More-detailed experiments

For more information:
ssasy@uwaterloo.ca oohrim@microsoft.com