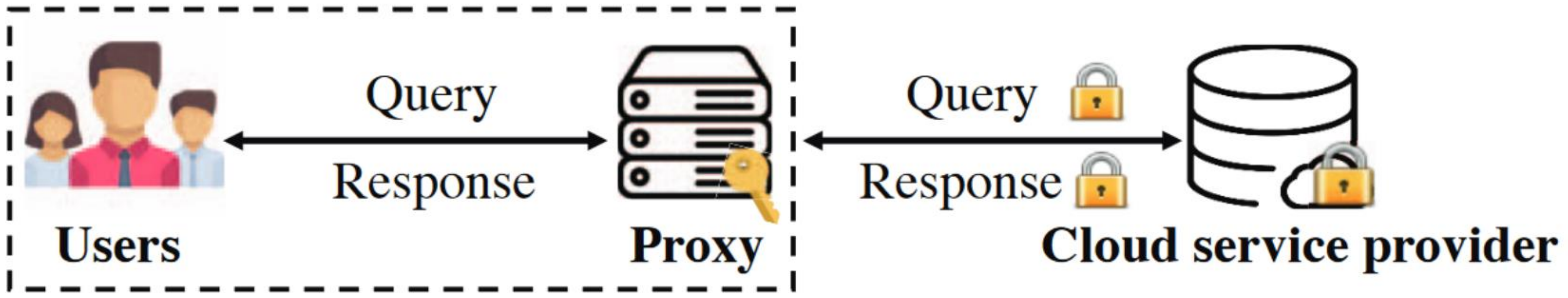
K-Indistinguishable Data Access for Encrypted Key-Value Stores

DESIGN OF A PRIVACY-PRESERVING KEY-VALUE STORAGE SYSTEM THAT CAN RESIST ACCESS PATTERN ATTACKS WITH MINIMAL STORAGE AND BANDWIDTH OVERHEAD

Problem Statement

System structure and threat model



Users access the data in the store through the proxy. The proxy resides in the same internal network as the users.

The proxy encrypts key-value pairs using a pseudo-random function and a symmetric encryption algorithm (with a secret key, that can not be compromised). Protection against length-leakage is provided by a padding of all the values to a same size.

The CSP provides data storage and query services.

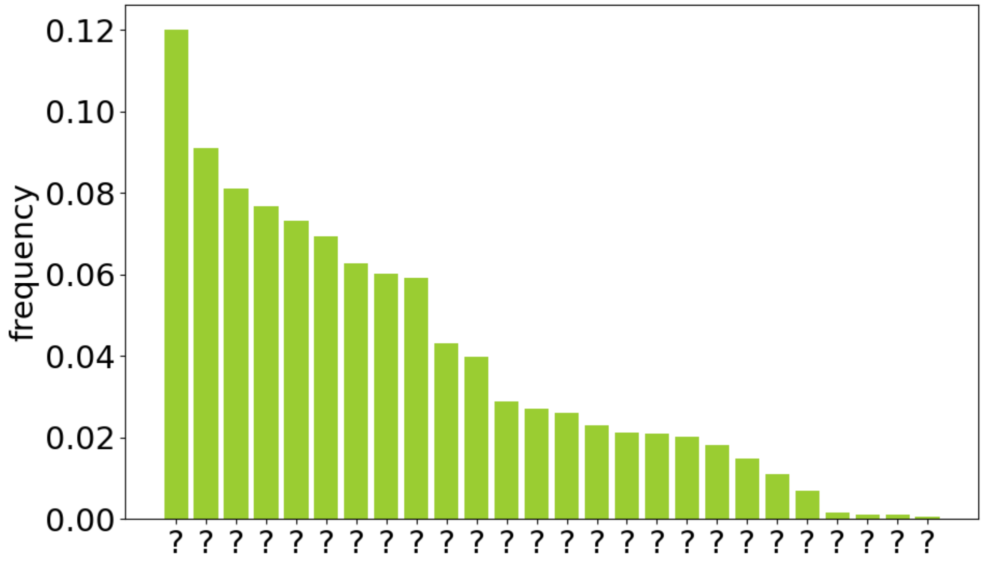
Adversaries can see the queries, but not understand them. They can not modify the queries or inject new queries.

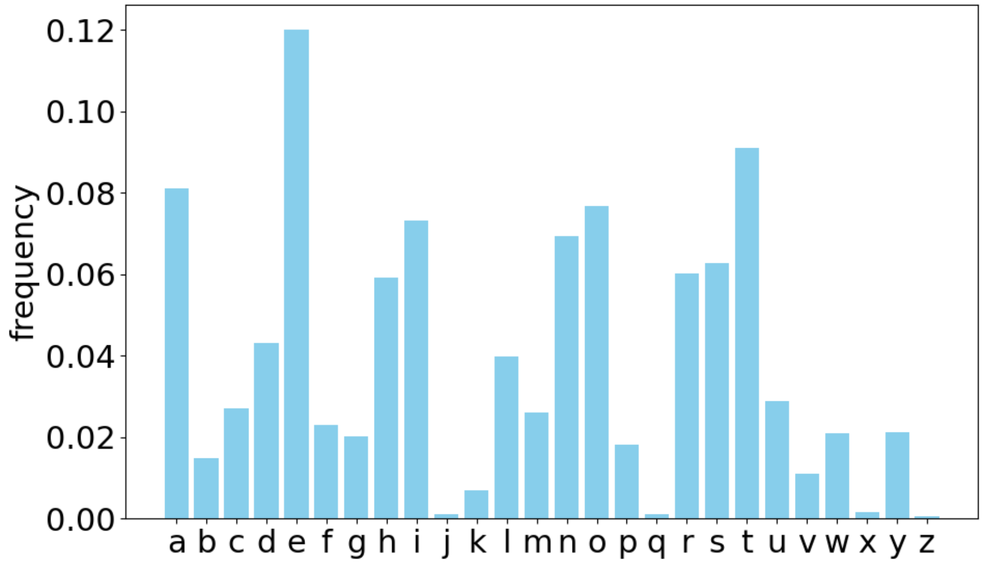
Access pattern attacks: frequency analysis

Passive persistent adversaries can launch access pattern attacks, such as frequency analysis.

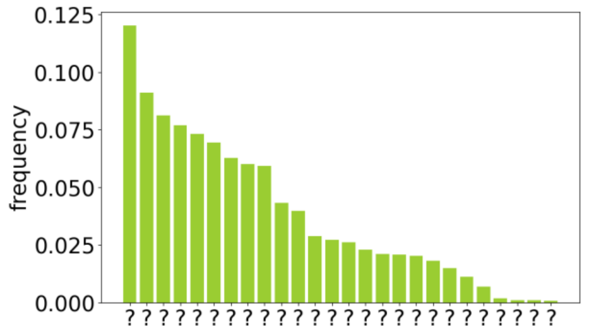
Adversaries can see access patterns, without understanding the content of the queries.

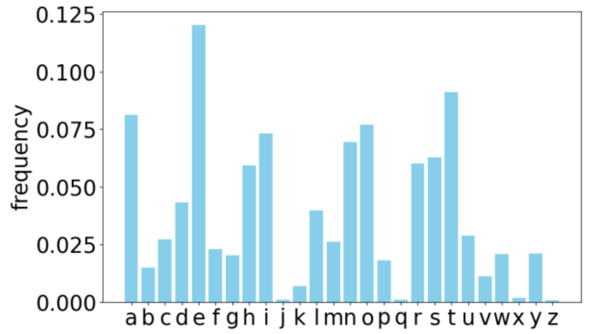
Adversaries have prior knowledge about the structure of the data stored.

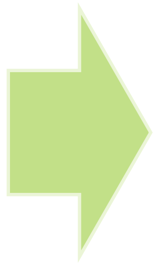




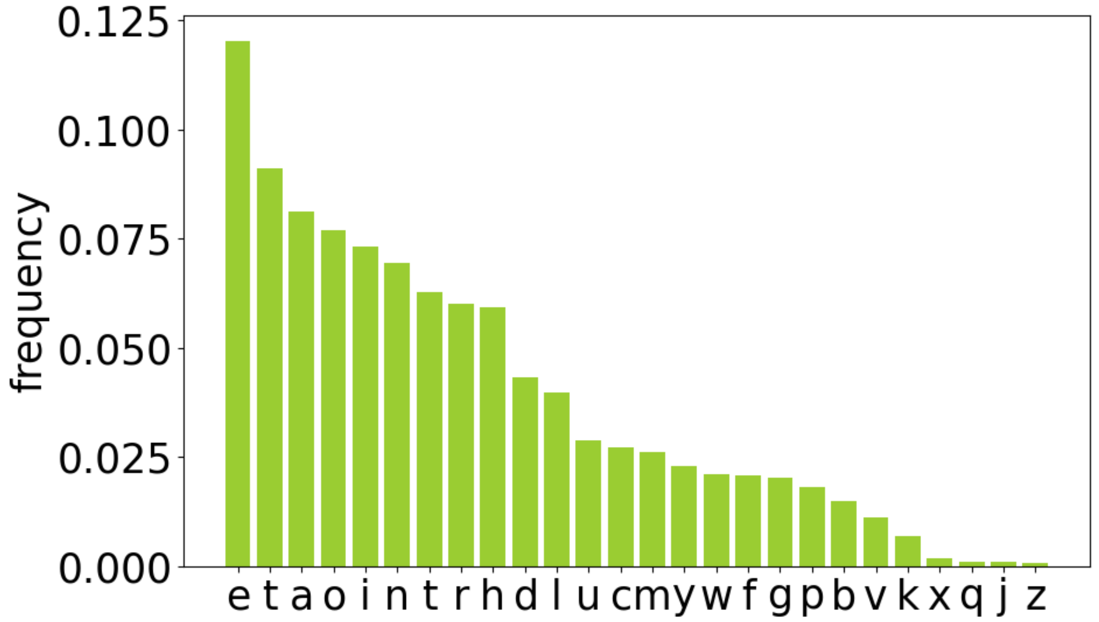
Access pattern attacks: frequency analysis







The key identities are discovered



Prior solutions: ORAM and pathORAM

ORAM: Oblivious Random Access machine.

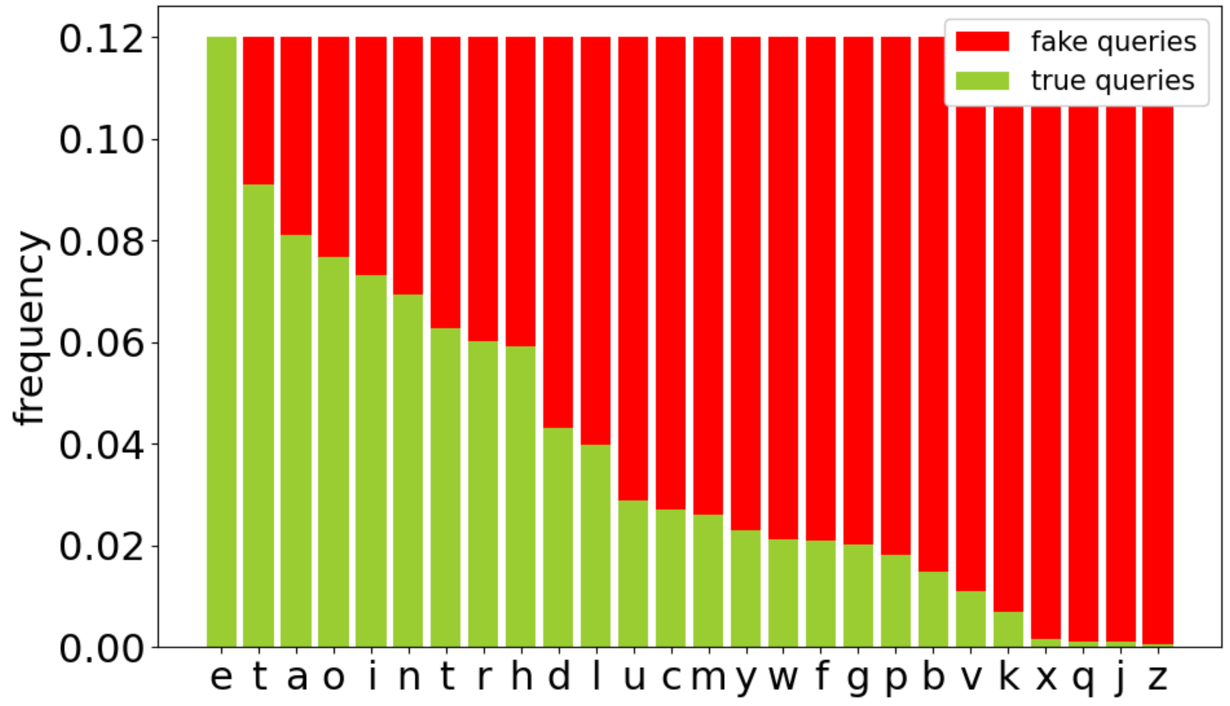
1. Given an element location/index on the database and an operation to perform on it, decide a subset of the remote storage that certainly (or at least with high probability) contains the desired element.
2. Download all the subset.
3. Decrypt locally the subset downloaded.
4. Find the desired element and perform the desired operation on it.
5. Shuffle the position of some or all the database elements in the subset.
6. Re-encrypt the subset with a randomized scheme, using the same symmetric key.
7. Re-upload the freshly re-encrypted data.

PathORAM

- Same concept of ORAM, but developed in a smarter way.
- The database is stored in a binary tree structure. Its map is stored locally.
- The subset that is downloaded in this case is a branch of the tree.

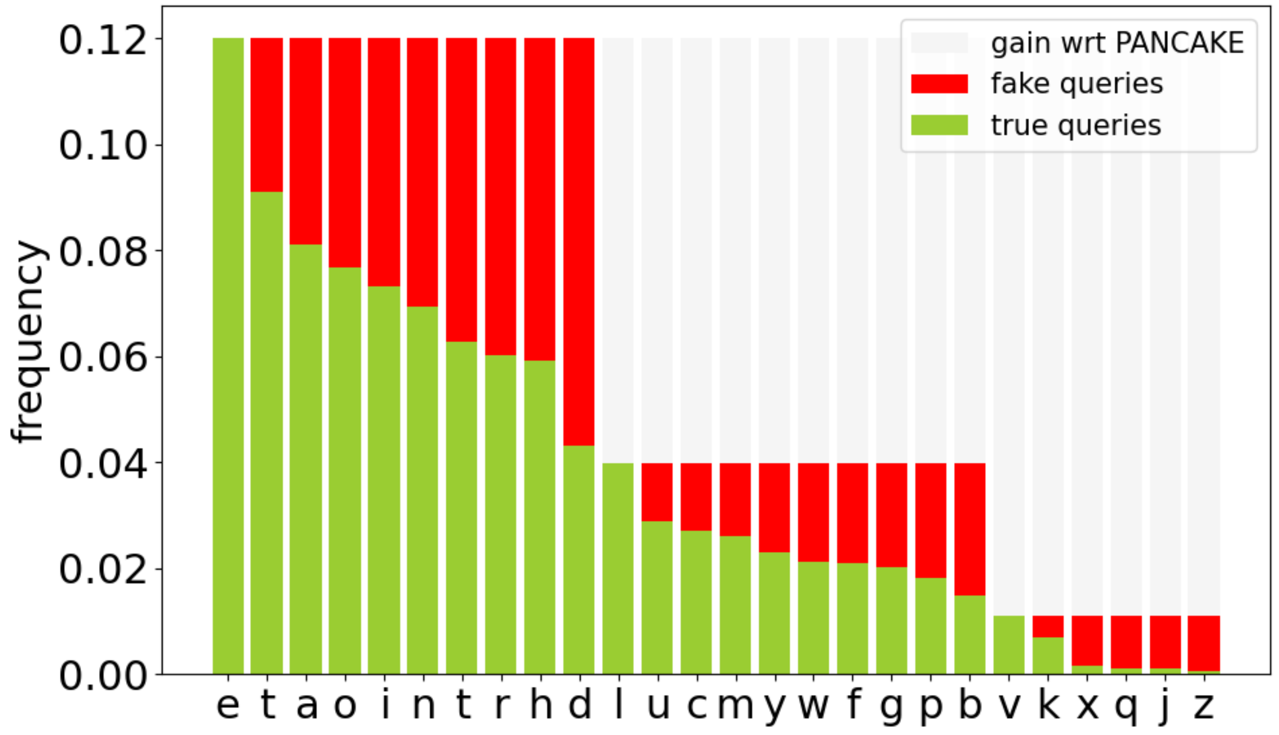
Prior solutions: PANCAKE

Frequency smoothing by adding fake queries: no pattern can be extracted from the access frequencies.



New solution: idea

The frequency smoothing is applied by intervals, depending on the frequency of subsets of keys.

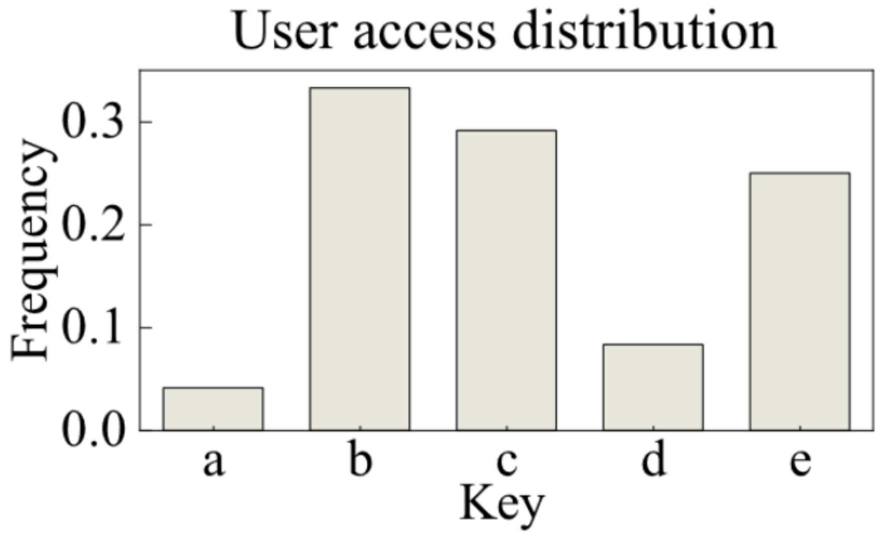


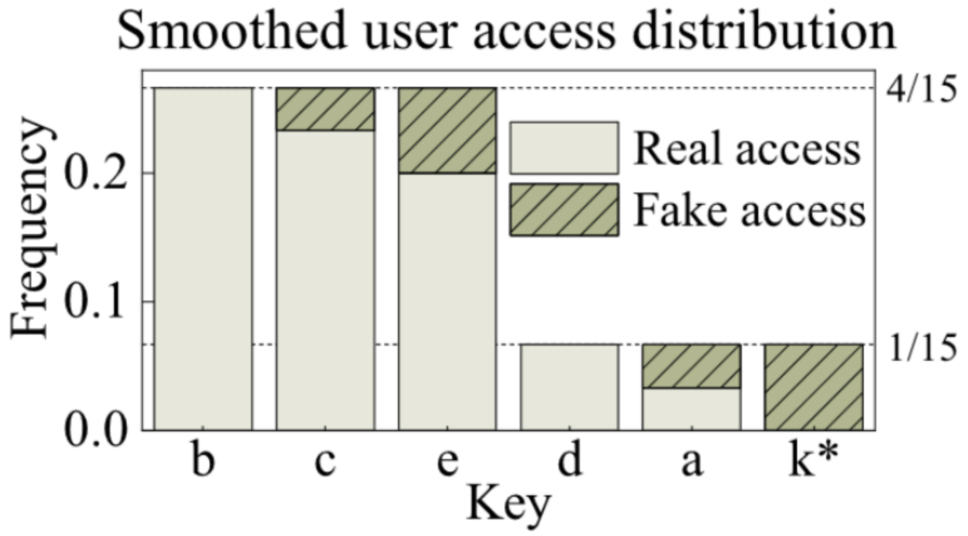
Design Implementation

K-indistinguishable Frequency Smoothing

Smooth the keys access frequencies into levels of K entries each.

The selection of the best K is critical to the system: trade-off between the bandwidth overhead and system security level.





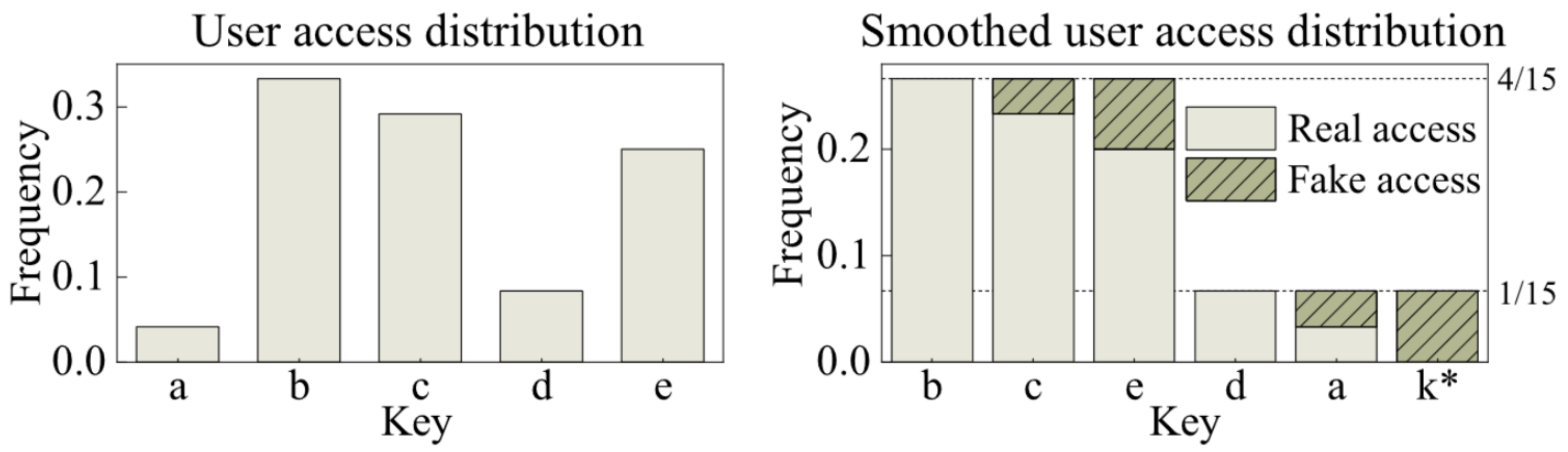
System initialization

The plaintext key-value store with is transformed into an encrypted key-value store and then uploaded to the CSP.

The proxy estimates the entries access frequencies. Then sets different frequency levels for the key-value store and assigns each entry into one of the levels based on its access frequency.

Dummy key-value pairs are added to the last level.

$$\alpha D_e + (1 - \alpha) D_f = D_K$$



Query execution

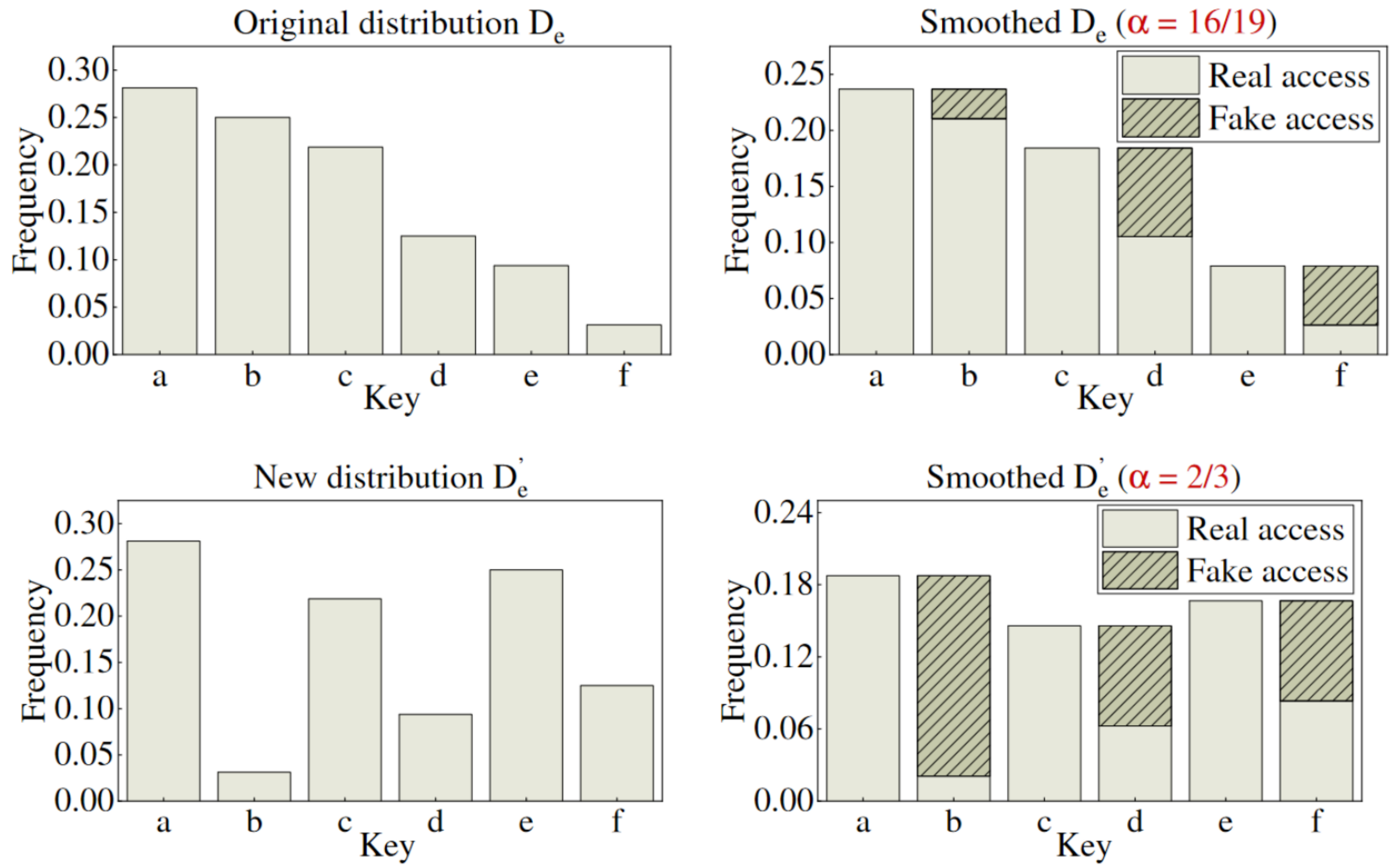
The proxy receives user queries and maintains a query queue, before routing them to the CSP.

For each request in the query, the proxy repeatedly flips a α-biased coin until it returns head, only then it routes the query to the CSP. For every tail a fake query is generated.

Problems arise if the stream of requests is not continuous: the last query is always real. There the necessity of a batch of fake queries at the end.

Write operations are problematic: the mechanism of fake queries can’t work. ORAM is used.

Dynamic Frequency Smoothing



Key swapping mechanism

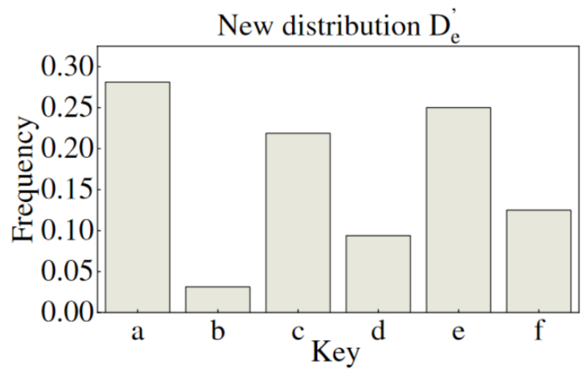
Due to the changes in the real queries frequency, the number of fake queries can increase dramatically.

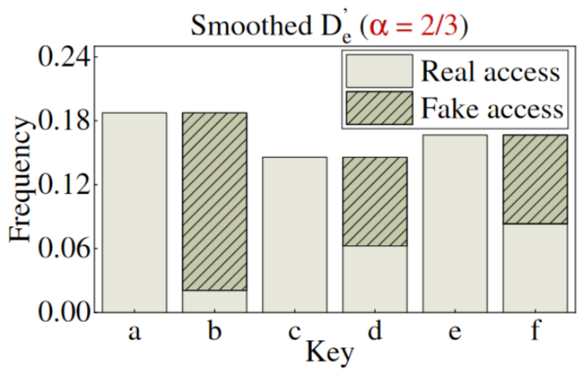
Downloading the whole database and repeating the initialization costs too much bandwidth and interrupts the service. An online adjustment method is needed.

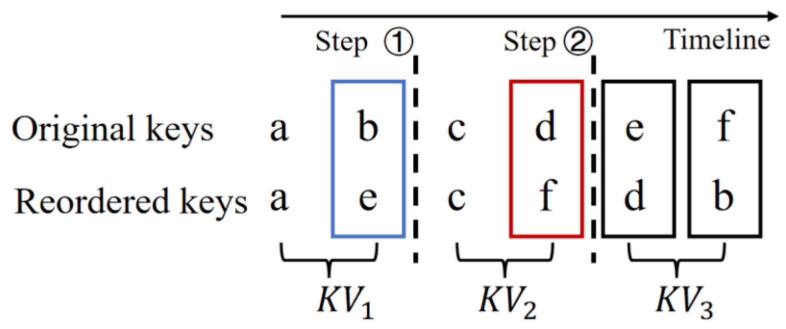
The main idea of key swapping is to reorder keys by access frequency and swap the keys whose access frequencies have a large difference from the rest of the keys in their respective groups (following the same logic used in the initialization). When one key leaves a group, there will be another key joining.

The pairs of keys to be swapped are found with an iterative algorithm, comparing the access frequencies between the different keys.

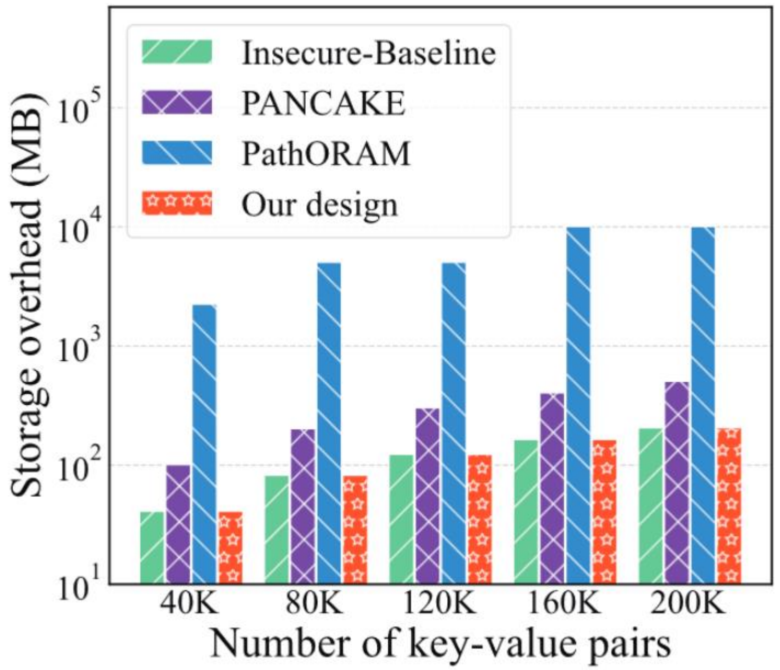
A threshold for swap operations is necessary, otherwise the swap operations will occur too frequently causing an increase in the overhead.

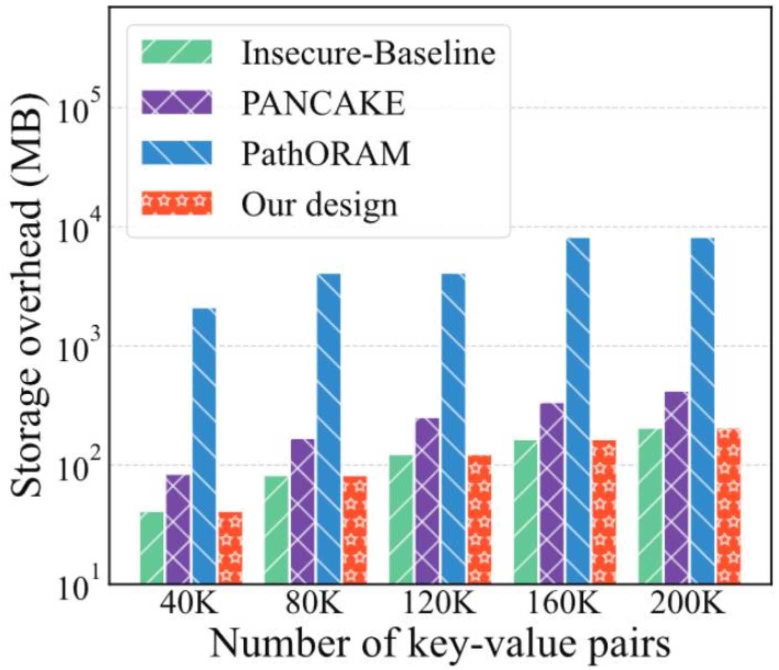




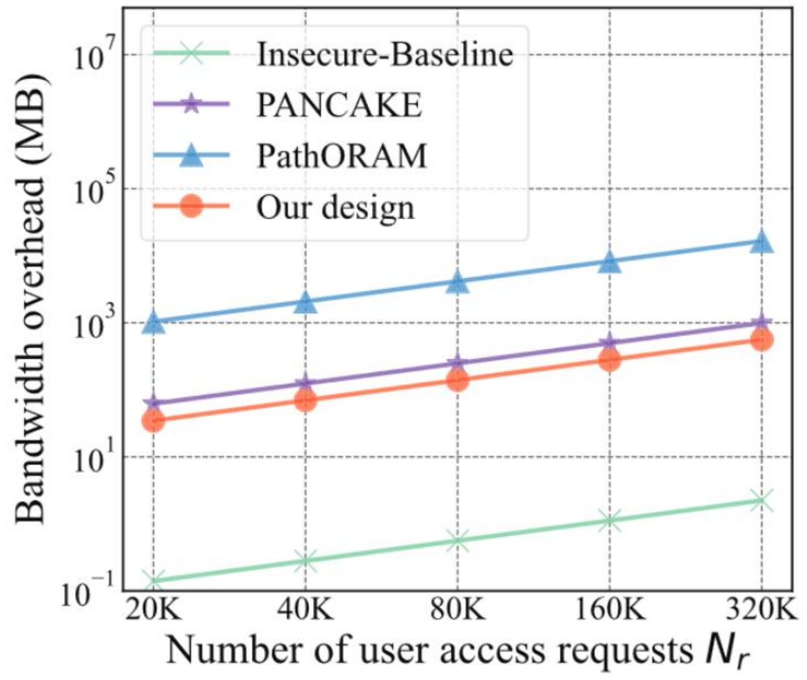


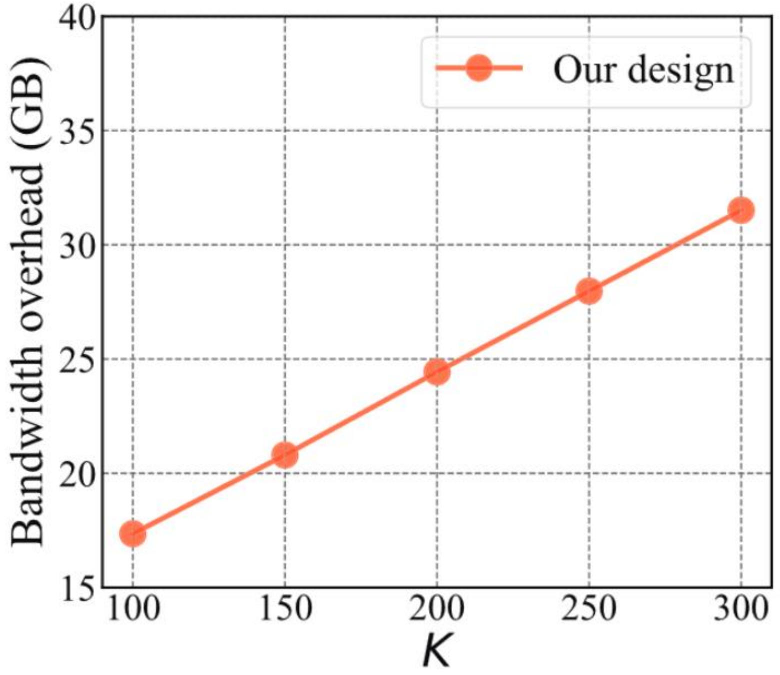
Storage overhead evaluation



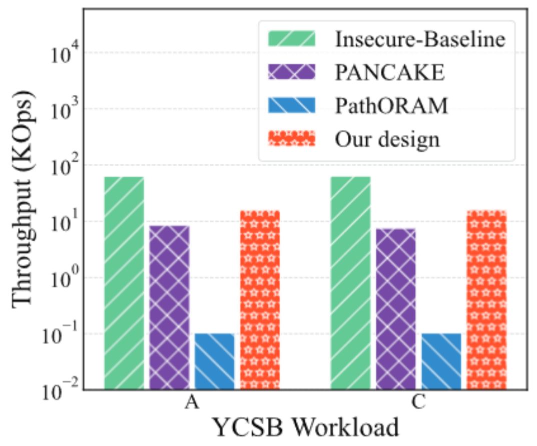


Bandwidth overhead evaluation

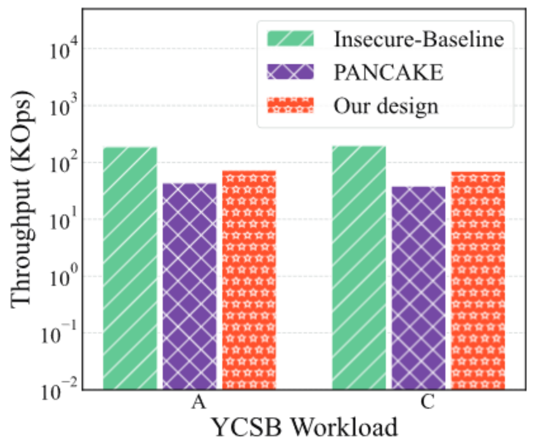




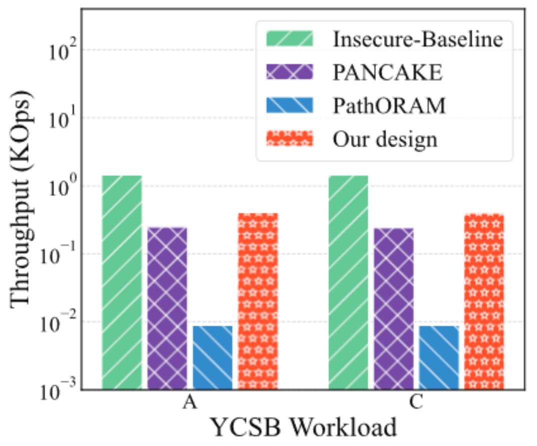
Throughput evaluation



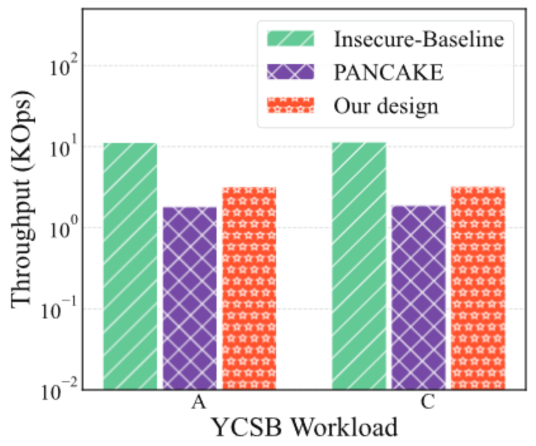
(a) Redis - single proxy thread



(b) Redis - multiple proxy thread



(c) RocksDB - single proxy thread



(d) RocksDB - multiple proxy thread

Paper evaluation

Originality:	2/5
Technical meaning:	5/5
Presentation:	5/5
Focus:	4/5
Overall:	4/5