Dynamic Searchable Encryption with Optimal Search in the Presence of Deletions

Javad Ghareh Chamani
HKUST

Dimitrios Papadopoulos
HKUST

Mohammadamin Karbasforushan
UCSC

Ioannis Demertzis
UCSC

香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

USENIX Security
August 10th – 12th, 2022

UC SANTA CRUZ

Encrypted Search

Motivation: Private Cloud Computing

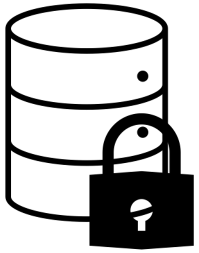
- Sensitive Data, GDPR, US HIPAA, …
- **Encryption** **+ Encryption key at client-side**
  + Efficient search and update

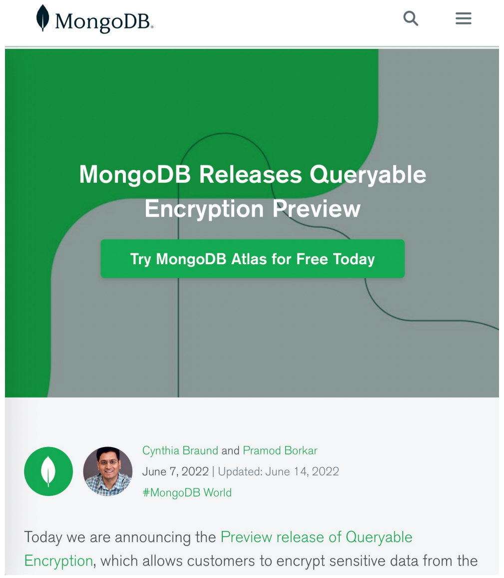
Searchable Encryption [SWP00] [DPPS20] [KM18] [KMO18] [PPYY19] [CJJKRM13] [CNR21] [CK10] [DPPDGP18] [DPPDG16] [FJKNRS15] [KM17] [MKNK15] [ANSS16] [DPP18] [DP17] [DTP18] [MM17] [BBFMR21] [HSWW18] [RMO15] [RMO18] [WP21] [YLLJC14]

- Especially, dynamic with efficient search [B16] [KP13] [KKPR12] [GPPR18] [CXWZSJ21] [DGPP20] [EKPE18] [KKLPK17]

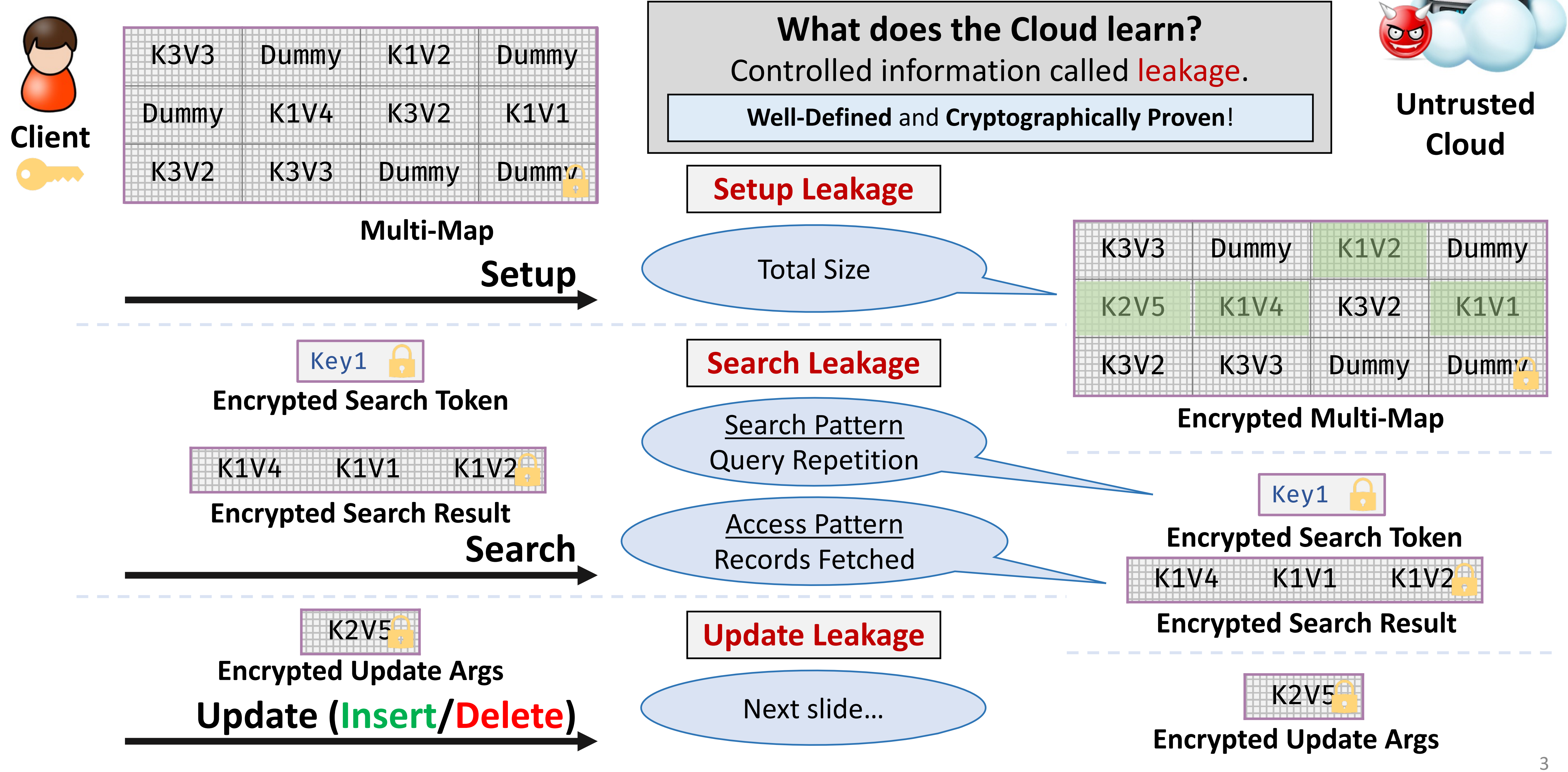
Applications:

- MongoDB's Queryable Encryption
- Pixek: Annotated Image Search
- Gun Registry (US Senate & Brown University)
- Encrypted key-value stores and databases
- Private end-to-end email communications with search capabilities
  - Via encrypted inverted index





Dynamic Searchable Encryption (DSE) [LSDHJ10] [SPS14] [BMO17]



DSE Update Leakage

Client

Untrusted Cloud

Search(Key1)

Insert(Key1, Val5)

Formally, the update should leak nothing about the key

Accessed memory locations during search

Forward Privacy [SPS14] [ZKP16]: The untrusted cloud should not link updates to previous searches.

… during updates, about searches

K3V3	Dummy	K1V2	Dummy
Dummy	K1V4	K3V2	K1V1
K3V2	K3V3	Dummy	Dummy

Encrypted Multi-Map

Backward Privacy [SPS14] [BMO17]: Controlled information about inserted-then-deleted values should be leaked. (at least not the value)

… during searches, about updates

Insert(Key3, Val6)

Delete(Key3, Val6)

Search(Key3)

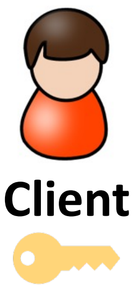
K3V3	K3V6	K1V2	Dummy
Dummy	K1V4	K3V2	K1V1
K3V2	K3V3	K3V6	Dummy

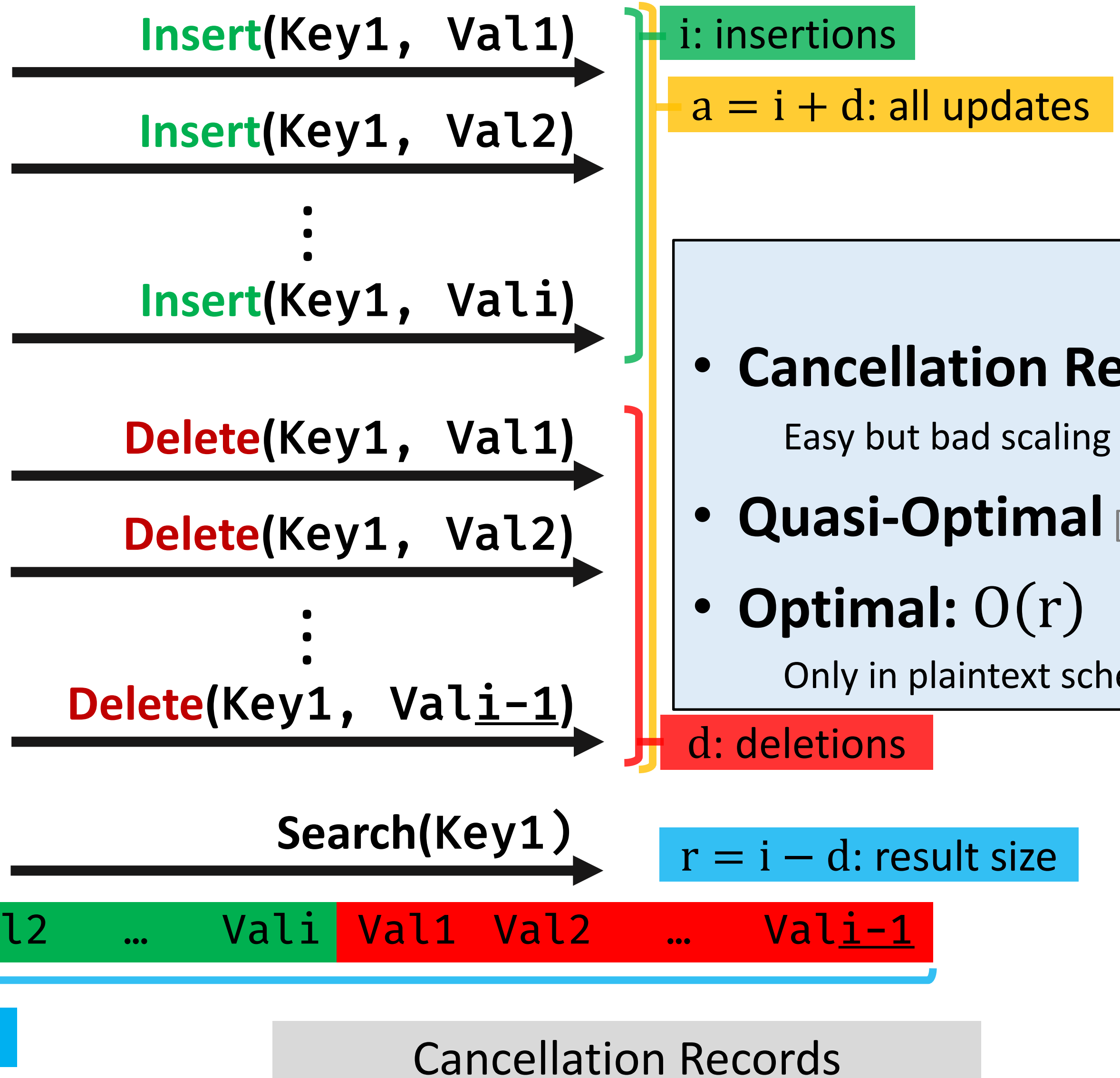
Encrypted Multi-Map

Forward and Backward Privacy have become the de-facto requirements in the literature.

Prior Work: Sub-Optimal Search

[GPPJ18] [DGPP20] [BMO17]



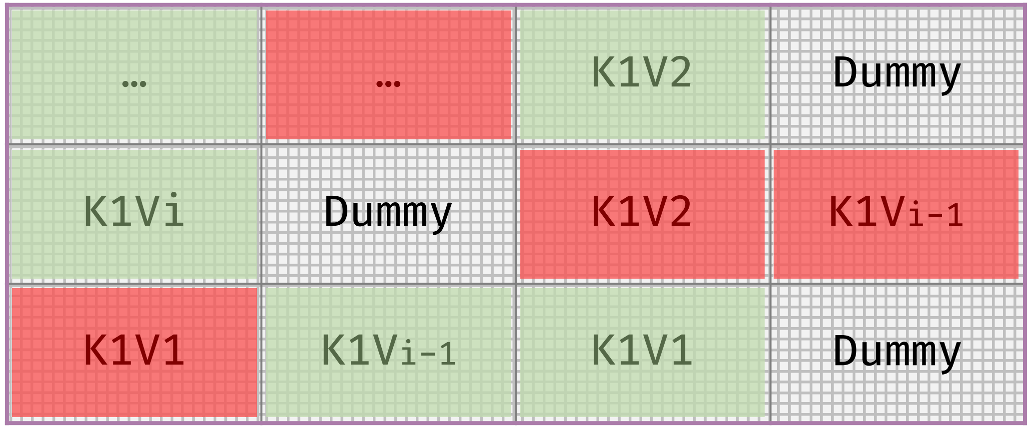


Search Time

- **Cancellation Records** [GPPJ18] [DGPP20] [BMO17]: $O(a)$

  Easy but bad scaling with deletions
- **Quasi-Optimal** [SPS14] [GPPJ18] [GPPJ18] [DGPP20]: $O(r \cdot \text{polylog}(N))$
- **Optimal:** $O(r)$

  Only in plaintext schemes!

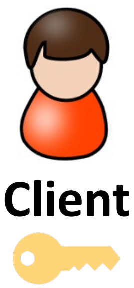


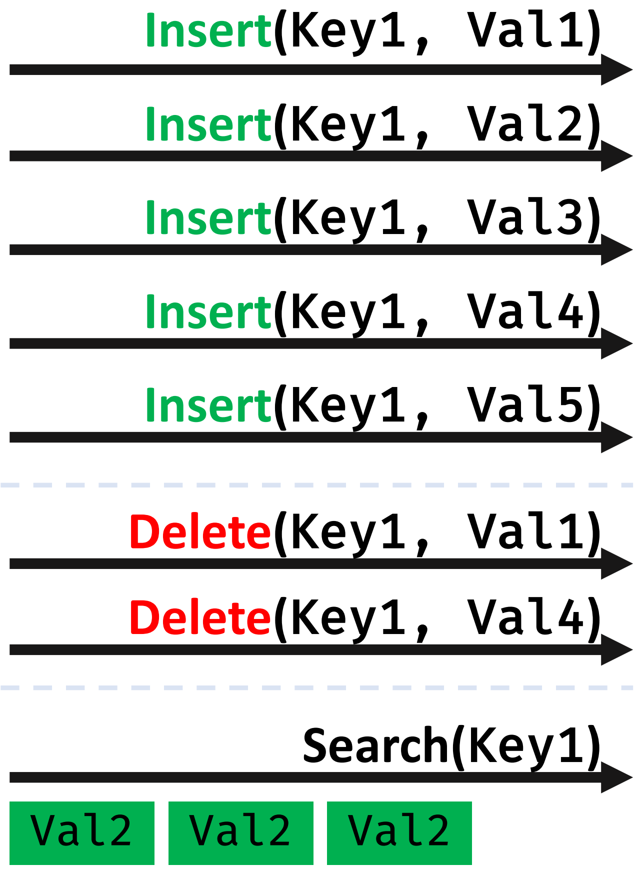
N: capacity

Cancellation Records

Easy way to stay **forward private**

Achieving Optimal Search with Deletions (OSSE)

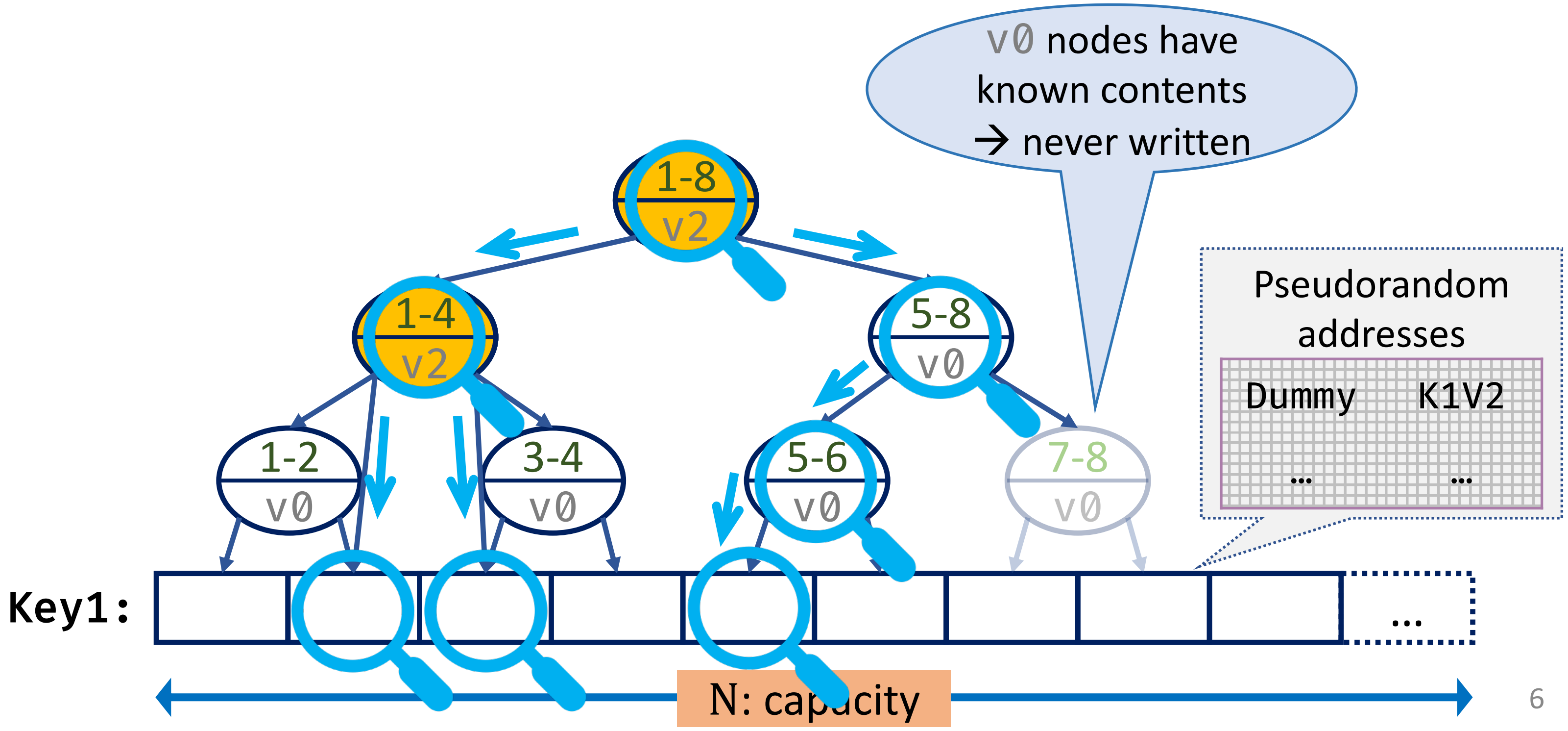




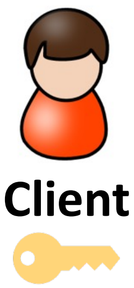
- Updates propagate up the tree
- $O(D \cdot \log N)$ space for deletions (D: total deletions)
  - Always write $\log N$ nodes to stay forward private
- Number of valid nodes remains $O(r)$…

Search Time = $\mathbf{O(r + \log i)}$

- Binary tree on top of entries to avoid deleted regions
- `vi`: node version, increased when pointers change



Optimal Search Details



Search(Key1)

Key1 🔒

Encrypted Search Token

@ H(H(🔑, Key1), id=1-8, ver=2)

@ H(H(🔑, Key1), id=1-4, ver=2)

Left: id=2, ver=0	Right: id=3, ver=0

Encrypted Search Token: points to the root

- Hash (PRF) of encryption key and search query + root id + root version
  → H(🔑, Key1), id=1-8, ver=2

Client needs root id and version of each key

→ Stores per-key insertion and deletion counters

→ Can use Oblivious Encrypted Map (OMap) [WNLCSSH14] for $O(1)$ client storage

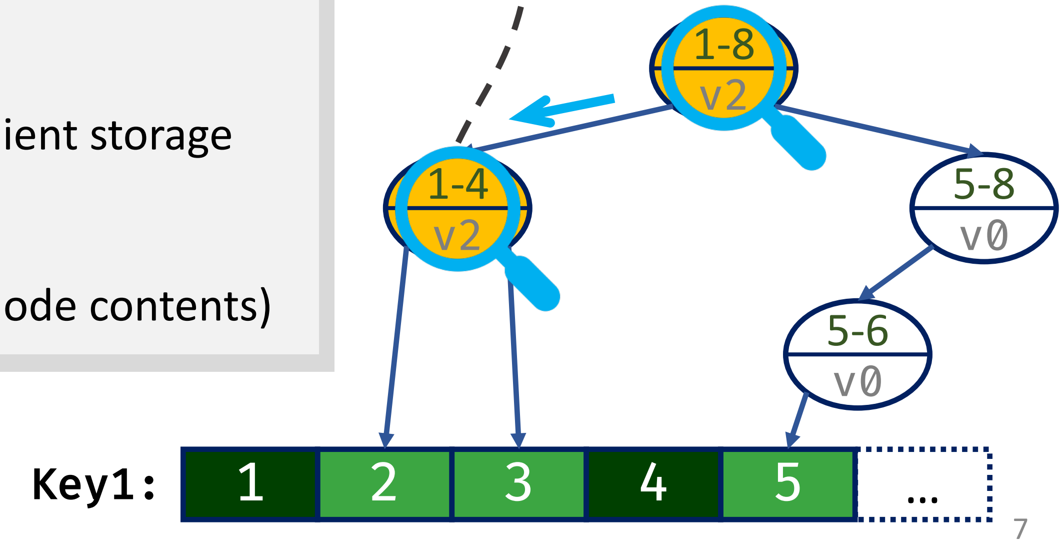
→ Hides op and args; adds $O(\log^2 N)$ cost over $O(\log N)$ rounds

Updates and Forward Privacy

→ OMaps to store tree state across updates (value-leaf mappings, node contents)

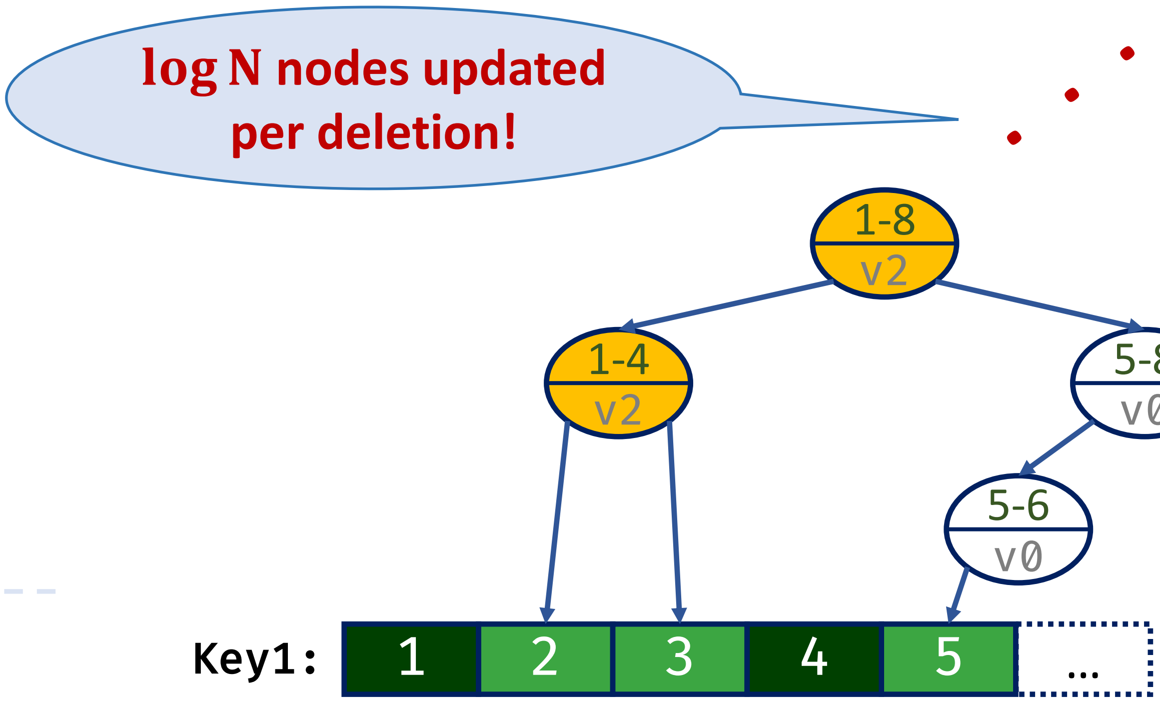
More costly updates to get optimal search

$O(N + D \cdot \log N)$ Space

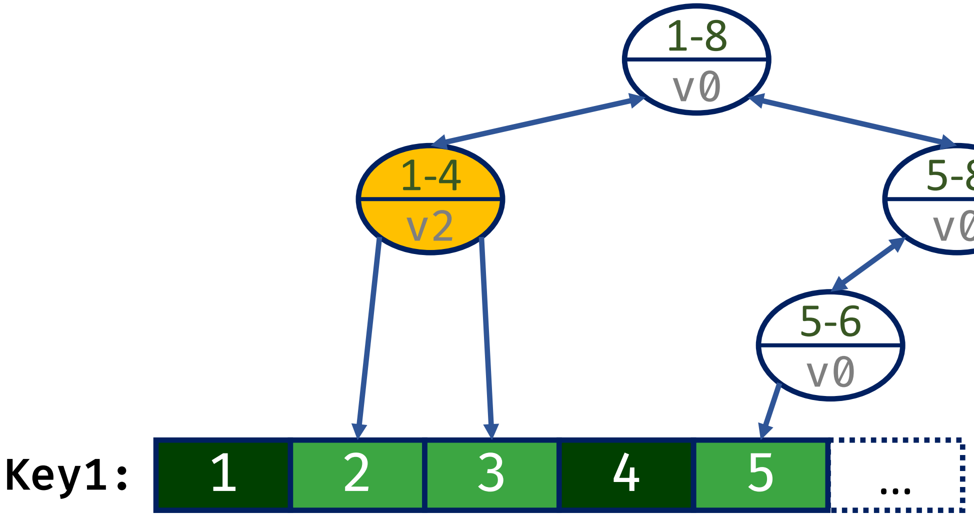


Second Scheme (LLSE)

- OSSE uses $O(N + D \cdot \log N)$ space
- How to prevent delete propagation?



- Idea: Don't store children's versions
- Binary search for latest node version
- Asymptotically and empirically faster than previous State-of-the-Art!



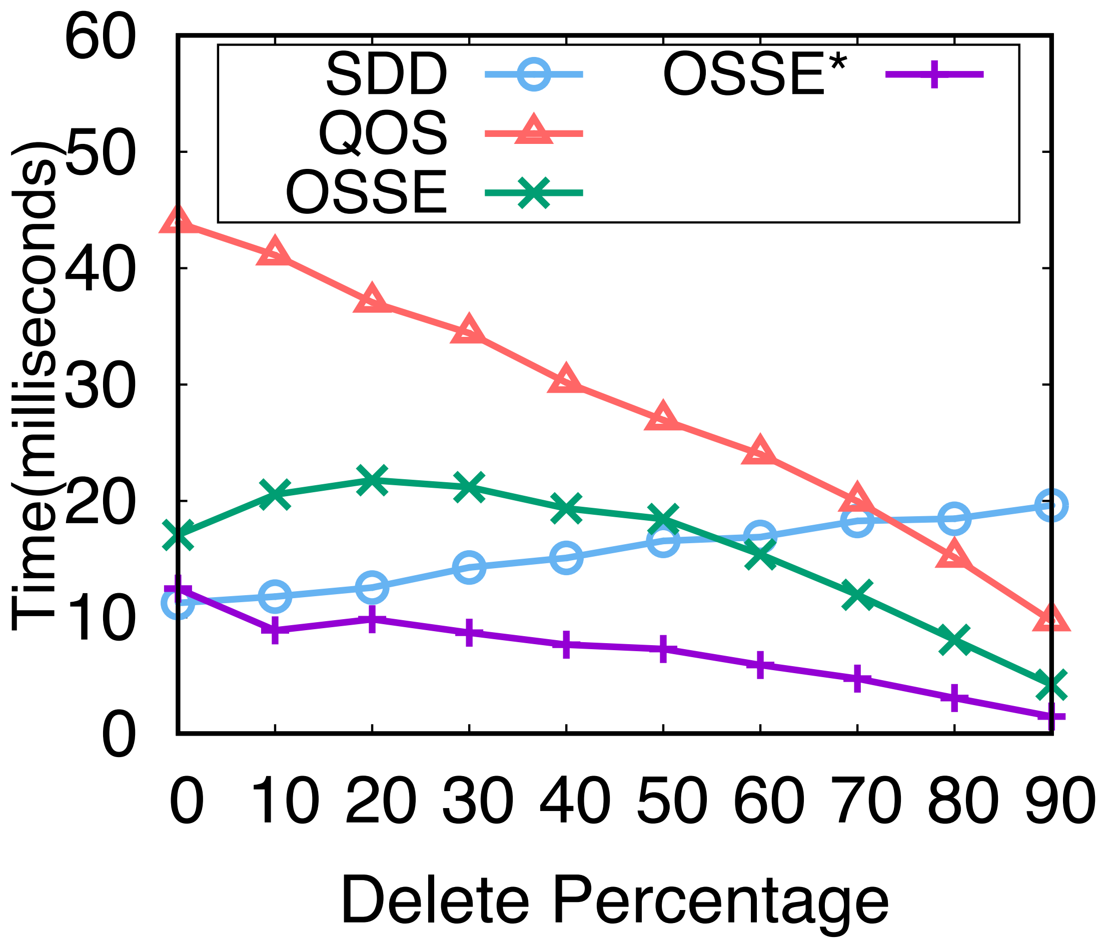
$O(\log N)$ **Deletion Speedup**
$O(N)$ **Space**
$O(r \cdot \log\log N)$ **Search**

Experiments

- Implemented in C++, using OpenSSL for crypto, AES-NI enabled
  - Open-source: github.com/jgharehchamani/OS-SSE

- compared with the best scheme with cancellation records (SDD) and previous SotA quasi-optimal scheme (QOS) [DGPP20]
- Hardware setup: 8-core Intel Xeon E-2174G 3.8Ghz, 128GB RAM, Ubuntu 16.04 LTS
- Experiments ran on a single machine
- Several optimizations compatible with privacy and leakage profile

Experiment: Search Time



Setup

- Search Time vs. Delete Percentage
  - DB Size = 1M, inserted values = 20K
  - Deleted uniformly chosen values
- SDD: Best cancellation records scheme
- QOS: Best quasi-optimal search scheme
  - Previous SotA
- OSSE: Optimal Search Scheme (ours)
  - OSSE*: OSSE + optimizations

Results

- OSSE beats SDD faster than QOS
- OSSE beats QOS right off the bat!
- Optimizations bring a very significant boost

Conclusion

- **OSSE: First forward and backward private dynamic searchable encryption scheme with optimal search**

- LLSE: Asymptotically and empirically faster than previous SotA with $O(N)$ space vs. $O(N + D \log N)$ in OSSE

- Open-source!

github.com/jgharehchamani/OS-SSE

Thank You!