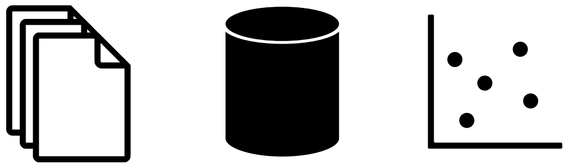
Injection-Secure Structured Encryption and Searchable Symmetric Encryption

Ghous Amjad, Seny Kamara,Tarik Moataz

Encrypted Search Algorithms

Trusted client

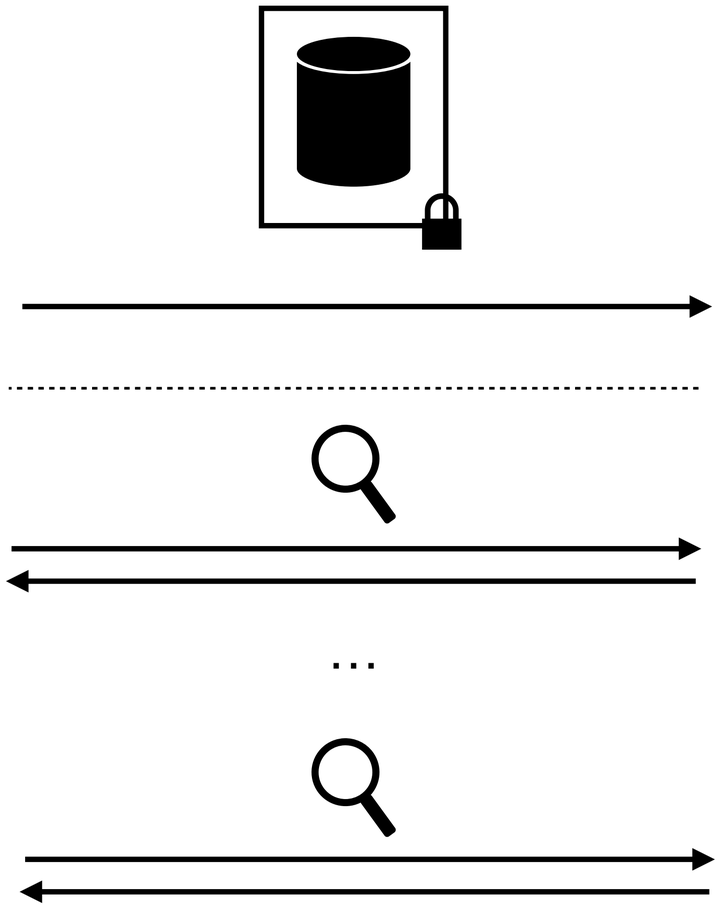


Untrusted server

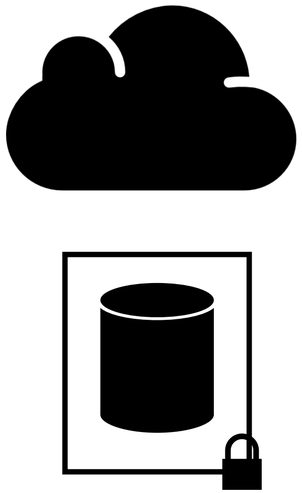
Encrypted Search Algorithms

Overview

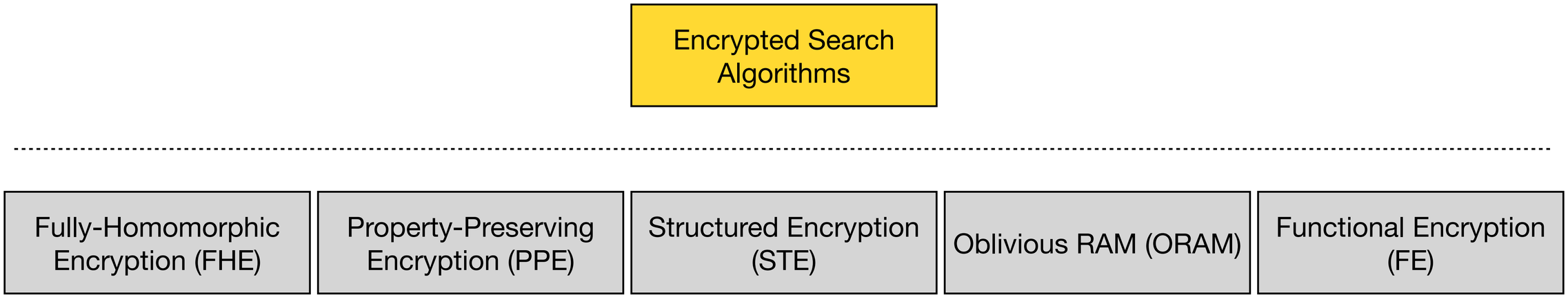
Trusted client



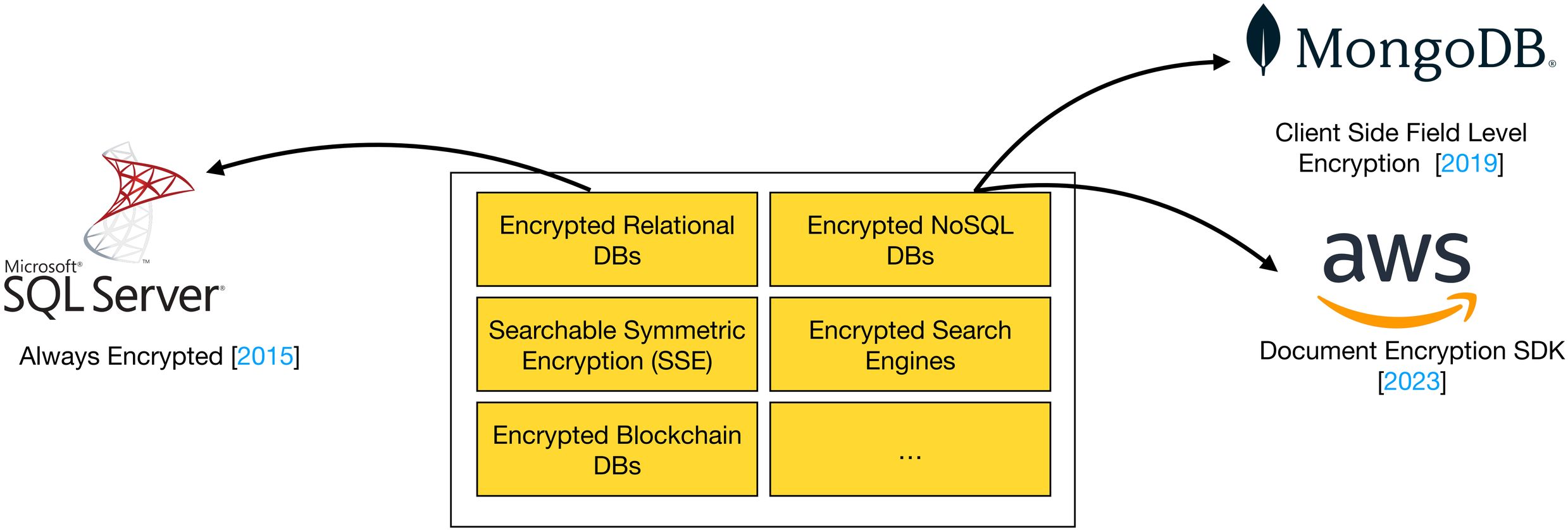
Untrusted server



Encrypted Search Algorithms

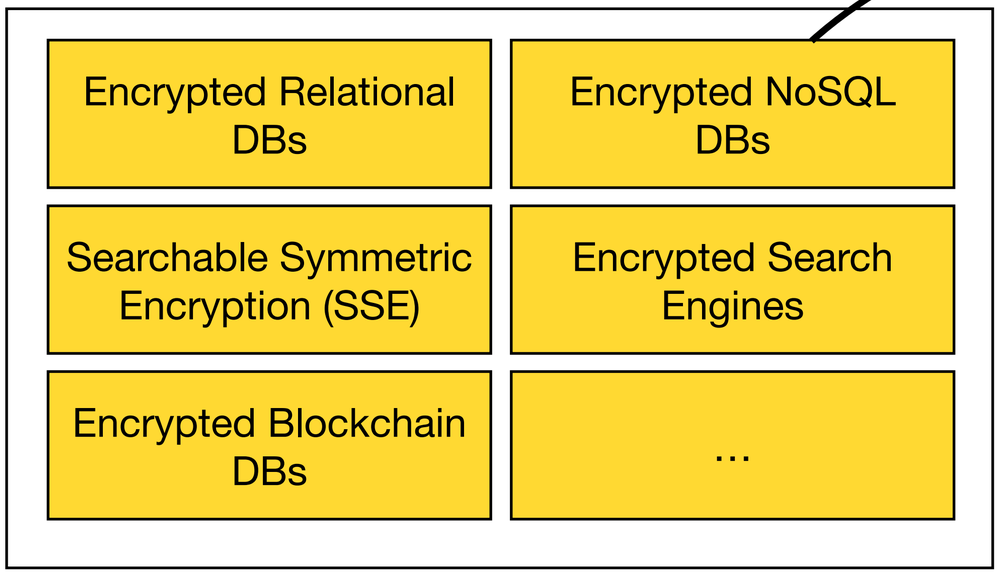


Encrypted Search Algorithms



Fully-Homomorphic Encryption (FHE) | **Property-Preserving Encryption (PPE)** | Structured Encryption (STE) | Oblivious RAM (ORAM) | Functional Encryption (FE)

Encrypted Search Algorithms



MongoDB®

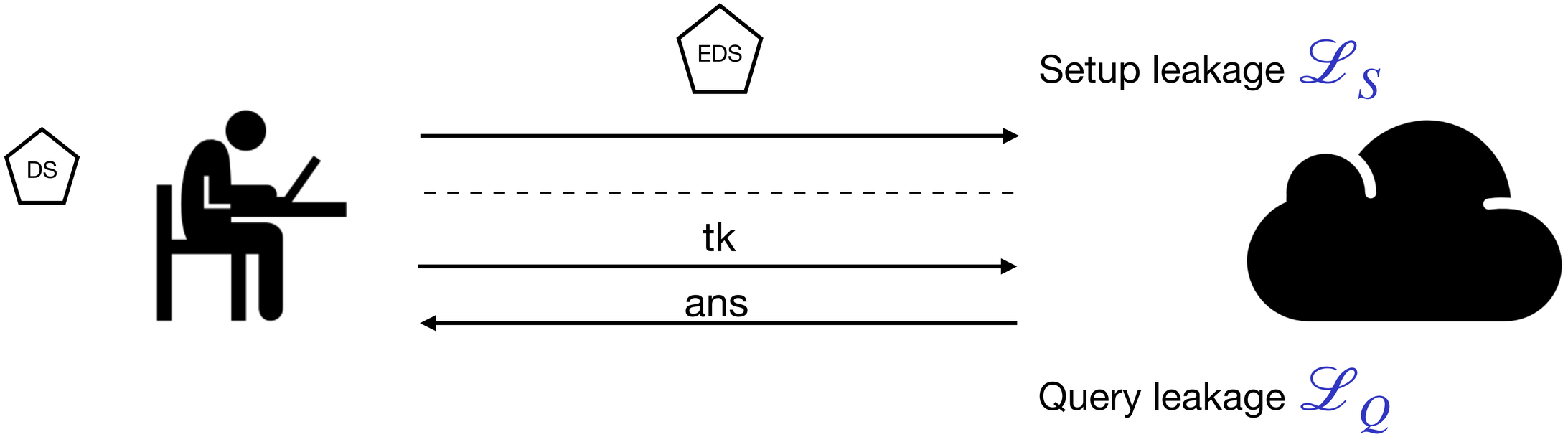
Queryable Encryption [2023]

Fully-Homomorphic Encryption (FHE) | Property-Preserving Encryption (PPE) | **Structured Encryption (STE)** | Oblivious RAM (ORAM) | Functional Encryption (FE)


STE

Structured Encryption

Definitions



$$[\text{key}, \text{EDS}] \leftarrow \text{Setup}[1^k, \text{DS}]$$

$$\text{tk} \leftarrow \text{Token}[\text{key}, q]$$

$$\text{ans} \leftarrow \text{Query}[\text{tk}, \text{EDS}]$$

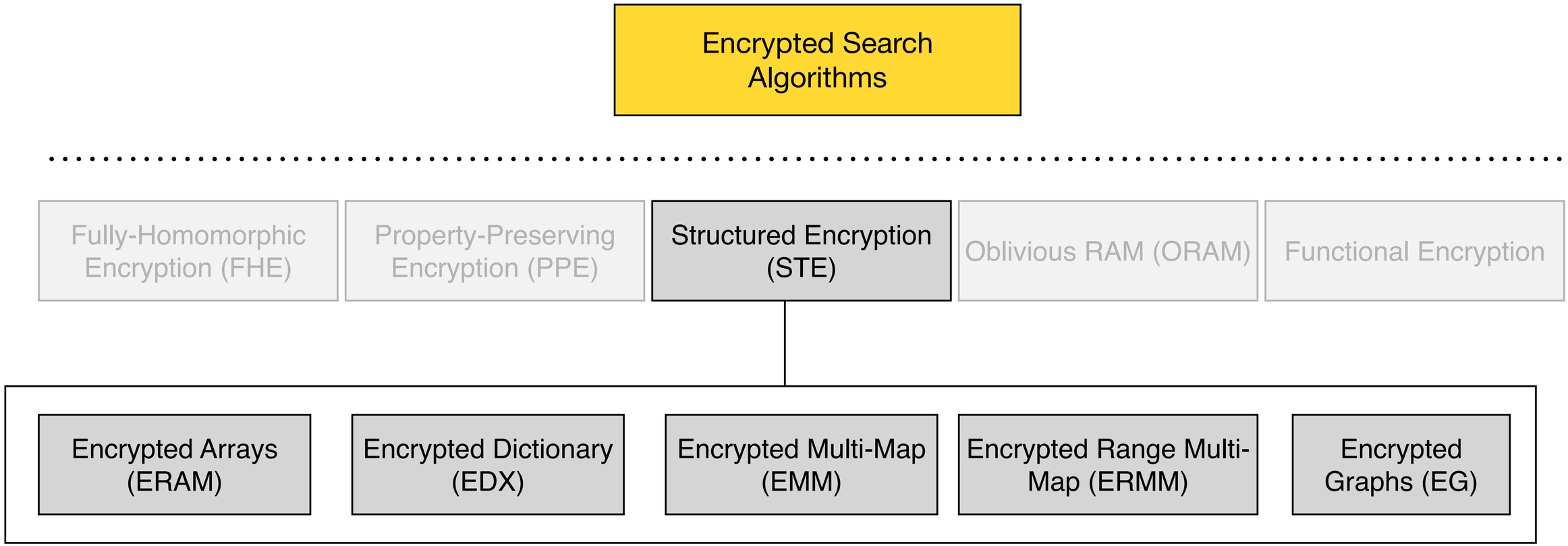
Structured Encryption

Definitions

- An STE scheme is $(\mathscr{L}_S, \mathscr{L}_Q, \mathscr{L}_A, \mathscr{L}_D)$-secure if
  - It reveals no information about the structure beyond $\mathscr{L}_S$
  - It reveals no information about the structure and queries beyond $\mathscr{L}_Q$
  - It reveals no information about the structure and adds beyond $\mathscr{L}_A$
  - It reveals no information about the structure and deletes beyond $\mathscr{L}_D$

Structured Encryption

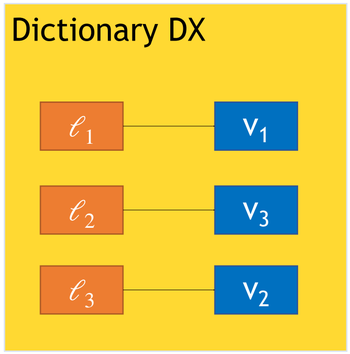
Data Structures



Background

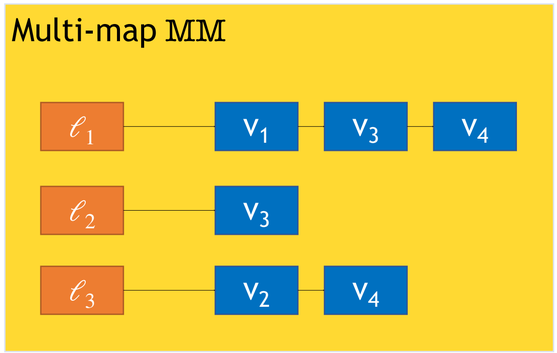
Dictionary and Multi-Map data structures

- DXs map labels to values



- Get: DX[ℓ_3] returns v_2

- MMs map labels to tuples

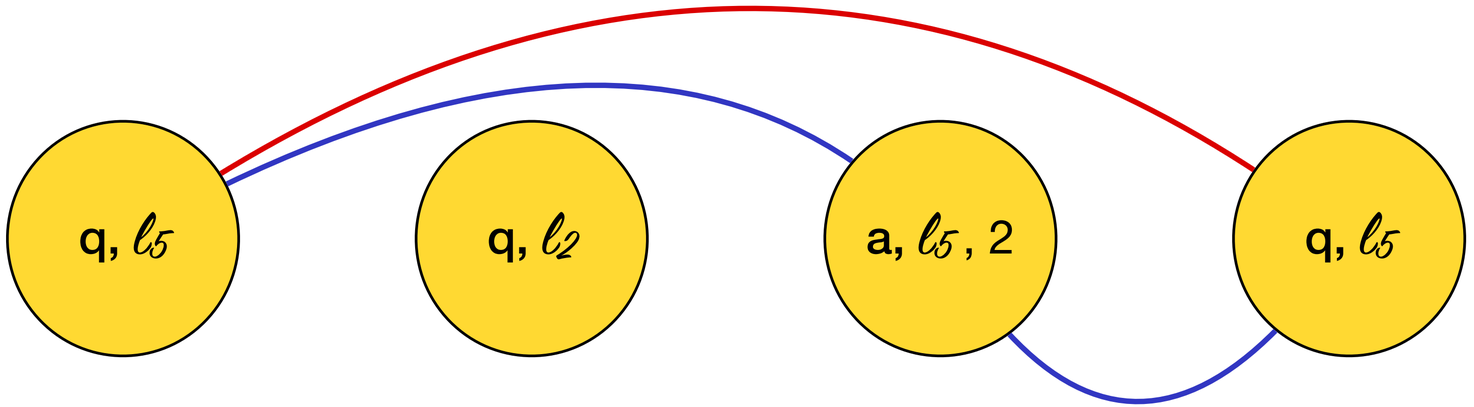


- Get: MM[ℓ_3] returns (v_2 , v_4)

Dynamic EMMs

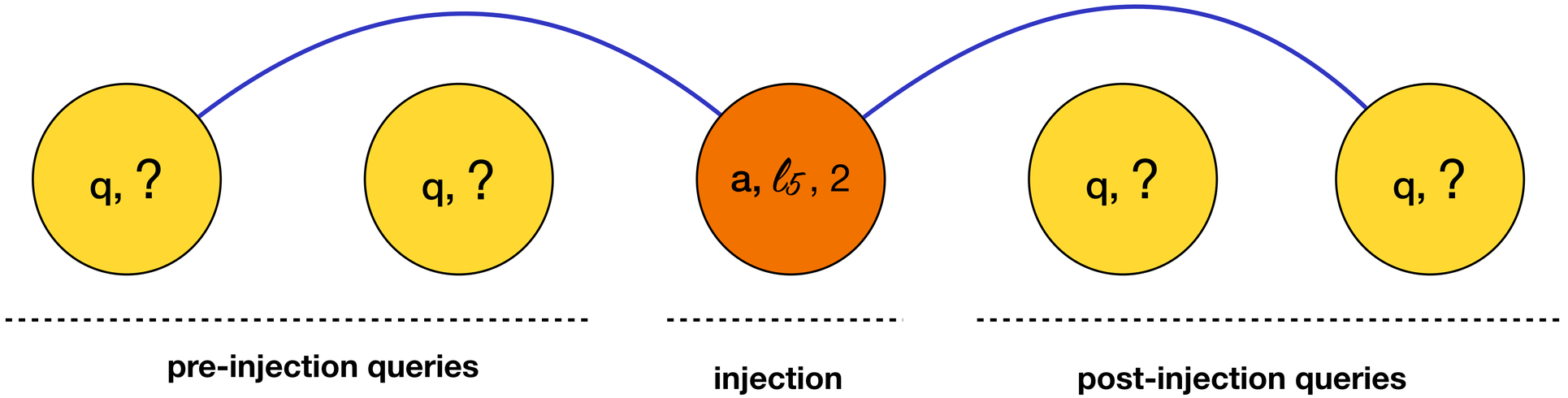
- Dynamic sub-linear EMMs [KPR12, CKKKRS14, …]
  - Optimal-time queries
  - $\mathscr{L}_Q$: query operations leak query equality (qeq)
    - If and when two queries are for the same label
  - $\mathscr{L}_A$: add operations leak add-query equality
    - If and when an add is for the same label as a query

Leakage Graphs



- $\mathscr{L}_A$: add operations leak add-query equality
- $\mathscr{L}_Q$: query operations leak query equality

Injection Attacks [ZKP16]



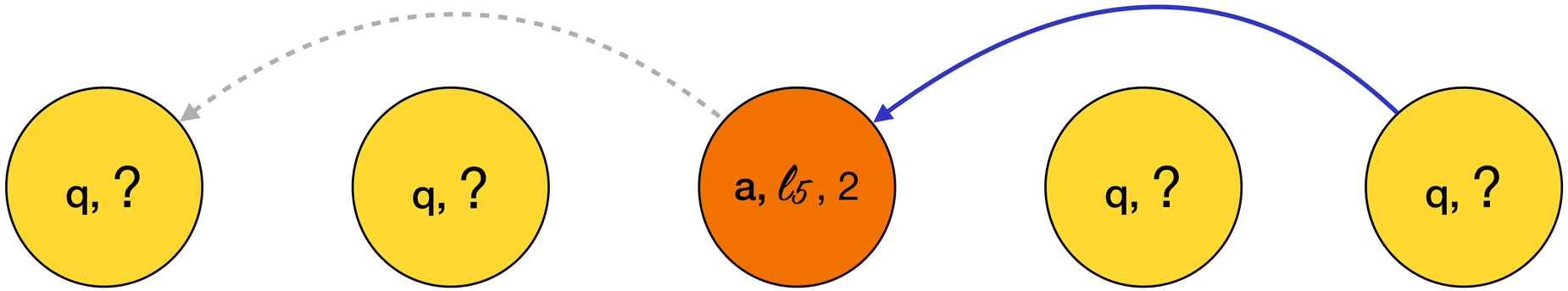
- $\mathscr{L}_A$: add operations leak add-query equality

Forward-Privacy [SPS14]

Definitions

- Introduced in [SPS14] but not formalized
  - “adds & deletes not correlated with previous queries”
- Formalized in [Bost16] roughly as “adds & deletes leak add & delete size”
- Formalized in [KM18] as “adds & deletes leak nothing”
- [ZKP16] observed that
  - Forward-privacy protects pre-injection queries
  - But not post-injection queries

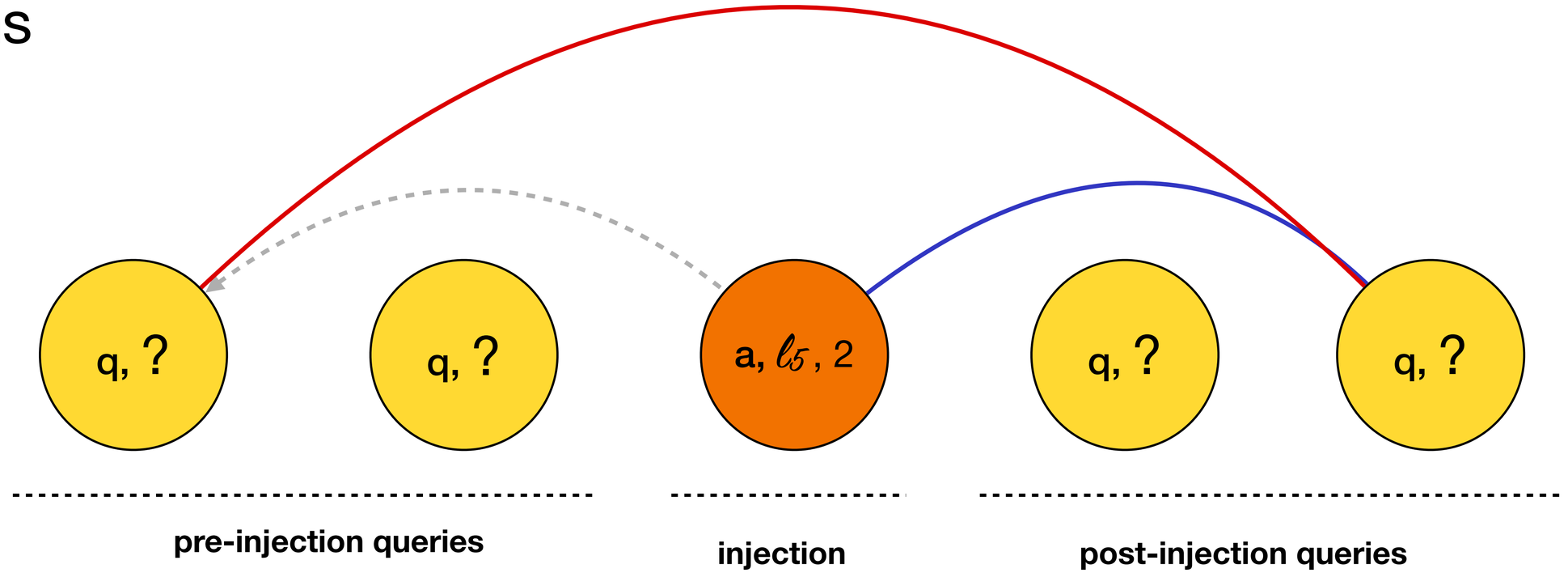
Injection Attacks [ZKP16]



- Forward-privacy
  - $\mathscr{L}_A$: add operations do not leak add-to-query equality but still leak query-to-add equality

Forward Privacy

Limitations



- But what if scheme *also* leaks query equality?
  - $\mathscr{L}_A$: add operations leak query-to-add equality
  - $\mathscr{L}_Q$: query operations leak query equality (qeq)

What’s Going on?

- Injection attacks are only a subset of a larger class of attacks

- Forward-privacy is only one security notion among a class of related notions

Correlation Attacks

- Learn labels of a subset of *operations*
  - using injections, inference attacks or known-data attacks etc…
- Use leakage to correlate known *operations* with unknown *operations*
- Operations here can be any of queries, adds, deletes, …

- Special case of injection attacks
  - execute *add* operations with known label
  - use *add-query* leakage to correlate adds to unknown *queries*

Correlation Security

- A leakage profile $(\mathscr{L}_S, \mathscr{L}_Q, \mathscr{L}_A, \mathscr{L}_D)$ is correlation-secure
  - “if it doesn’t reveal correlations between certain types of operations”
  - “if the leakage graph has no paths between certain types of operations”
  - Formalized using a (complex) parametrizable game-based definition

Injection Security

- Special case of correlation security
- A leakage profile $(\mathscr{L}_S, \mathscr{L}_Q, \mathscr{L}_A, \mathscr{L}_D)$ is injection-secure
  - “if there are no correlations between adds and pre- & post-injection queries”
  - “if no paths exist between adds and pre- and post-injection queries”
  - Formalized by parameterizing the definition of correlation security

Other Approaches

- ORAM-based
  - Store multi-map in an ORAM
  - but in a volume-hiding manner (see paper for more)
- SWiSSSE [GPPW20]
  - argument of security against injection attacks

Contributions

Theorem 1

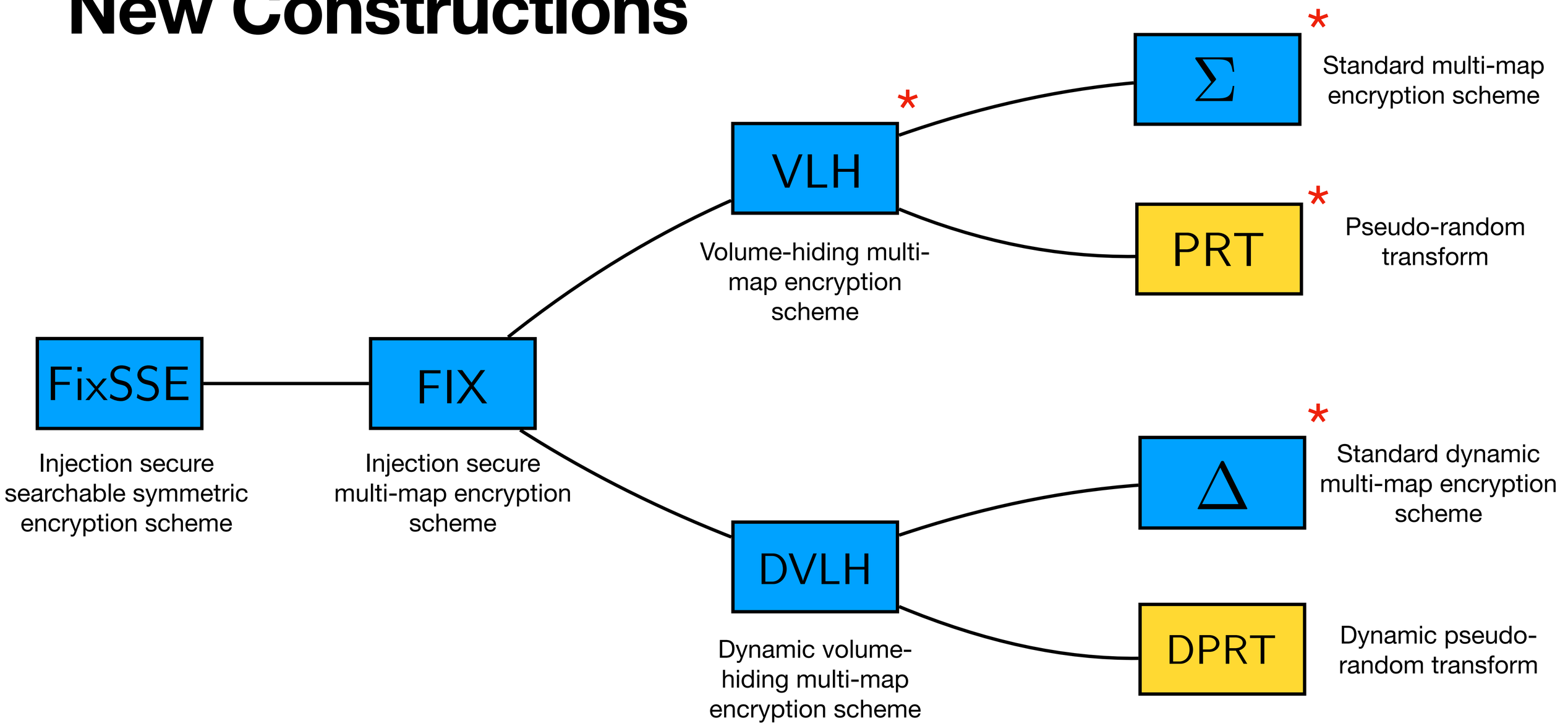
Theorem 1. If a leakage profile is such that $\mathscr{L}_Q = \mathrm{qeq}$ and $\mathscr{L}_A = \mathscr{L}_D = \perp$ then it is injection-secure.

New Constructions

FIX, FixSSE and DVLH

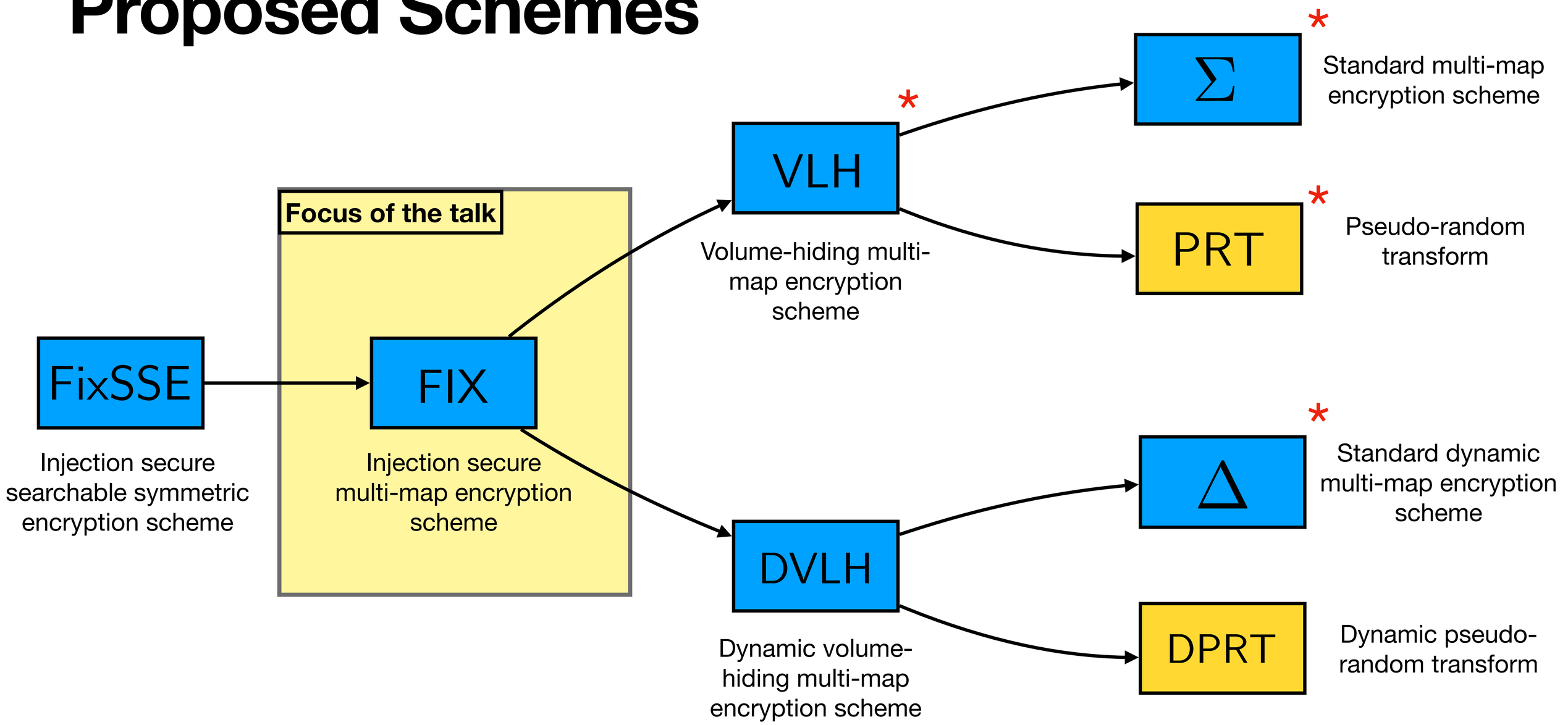
- FIX is the first (dynamic) injection-secure multi-map encryption scheme
- FixSSE is the first (dynamic) injection-secure searchable symmetric encryption scheme
- DVLH is a dynamic volume-hiding multi-map encryption scheme
  - DVLH is based on the dynamic pseudo-random transform (DPRT)
- All schemes above can achieve sub-linear search overhead under some assumptions

New Constructions



* existing schemes from the literature

Proposed Schemes



* existing schemes from the literature

Design Techniques

Randomized but Fixed Schedule

- Adds, deletes etc are stored in local stash…
  - …and pushed to server-side encrypted structures following a randomly sampled but fixed schedule
  - Scheduled defined by a random permutation π
- Every label ℓ has a scheduled push time/epoch defined by $\pi(\ell)$
  - Add, erase or edit tokens for ℓ are stored in stash until ℓ's scheduled push epoch
  - Once we reach the last epoch, we go back to the first one
- The client-side state can grow as a function of the distribution of adds, deletes etc.

FIX

Building Blocks

- A static volume-hiding multi-map encryption scheme
  - Can be instantiated using VLH [KM19], AVLH [KM19] or dprfMM [PPYY19]
    - In this talk we use VLH as the underlying instantiation.
  - Hides volume (response length) and reveals query equality
- A dynamic volume-hiding multi-map encryption scheme
  - Can be instantiated using the DVLH (*this work*) or 2ch [APPYY23]
    - In this talk, we use DVLH as the underlying instantiation.
  - Hides volume and reveals the query equality

FIX

Setup

$\text{Setup}\left[1^k, \text{MM}\right]$:

1. Sample a random permutation π
2. Create two empty multi-map MM_n and MM_{st} and a counter cnt
3. Compute

$$\left[K_o, \text{st}_o, \text{EMM}_o\right] \leftarrow \text{VLH.Setup}\left[1^k, \text{MM}\right]$$

$$\left[K_n, \text{st}_n, \text{EMM}_n\right] \leftarrow \text{DVLH.Setup}\left[1^k, \text{MM}_n\right]$$

4. The client receives $K = (K_o, K_n)$ and $\text{st} = \left[\text{st}_o, \text{st}_n, \text{MM}_{st}, \text{cnt}, \pi\right]$

   The server receives $\text{EMM} = \left[\text{EMM}_o, \text{EMM}_n\right]$

FIX

Append

Append $\left[K, \text{st}, \ell, \bar{v}; \text{EMM}\right]$:

1. For all v in $\bar{v}$, add **v || add** to $\text{MM}_{\text{st}}[\ell]$

2. Compute

$$\left[\text{st}'; \text{EMM}'\right] \leftarrow \text{Push}\left[K, \text{st}, \mu; \text{EMM}\right]$$

3. Client outputs the updated state st'

   Server outputs the updated multi-map EMM'

FIX

Erase

Erase $\left[K, \text{st}, \ell, \bar{v}; \text{EMM}\right]$:

1. For all v in $\bar{v}$,
   - if $\text{v} \,||\, \text{add} \in \text{MM}_{\text{st}}[\ell]$,
     - Remove $\text{v} \,||\, \text{add}$ from $\text{MM}_{\text{st}}[\ell]$
   - Else
     - Add $\text{v} \,||\, \text{del}$ to $\text{MM}_{\text{st}}[\ell]$
2. Compute

$$\left[\text{st'}; \text{EMM'}\right] \leftarrow \text{Push} \left[K, \text{st}, \mu; \text{EMM}\right]$$

3. Client outputs the updated state st'

   Server outputs the updated multi-map EMM'

FIX

Get

$\mathsf{Get}\left[K, \mathsf{st}, \ell; \mathsf{EMM}\right]$:

1. Initialize an empty set Result

2. The client and server execute

$$\left[R_o; \perp\right] \leftarrow \mathsf{VLH.Get}\left[K_o, \mathsf{st}_o, \ell; \mathsf{EMM}_o\right]$$

$$\left[R_n; \perp\right] \leftarrow \mathsf{DVLH.Get}\left[K_n, \mathsf{st}_n, \ell; \mathsf{EMM}_n\right]$$

3. The client computes the following sets

$$R_{st}^{+} = \left[v : v \| \mathsf{add} \in \mathsf{MM}_{st}[\ell]\right] \quad R_{st}^{-} = \left[v : v \| \mathsf{del} \in \mathsf{MM}_{st}[\ell]\right] \quad R_0^{-} = \left[v : v \| \mathsf{del} \| \mathsf{old} \in \mathsf{MM}_{st}[\ell]\right]$$

4. The client outputs

$$\mathsf{Result} = \left[R_o \bigcup R_n \bigcup R_{st}^{+}\right] \setminus \left[R_{st}^{-} \bigcup R_0^{-}\right]$$

FIX

Push (Part 1)

Push $\left[K, \text{st}, \mu ; \text{EMM}\right]$:

1. For all i in {1, … , μ },
   - Compute $j = \text{cnt} + i \mod \#\mathbb{L}_{\text{MM}}$
   - Compute

$$\left[R_o ; \perp\right] \leftarrow \text{VLH.Get}\left[K_o, \text{st}_o, \ell_{\pi(j)} ; \text{EMM}_o\right]$$

$$\left[R_n ; \perp\right] \leftarrow \text{DVLH.Get}\left[K_n, \text{st}_n, \ell_{\pi(j)} ; \text{EMM}_n\right]$$

   - For all $v \,||\, \text{flag} \in \text{MM}_{\text{st}}[l]$,
     - If flag = add, $R_{st}^{+} := R_{st}^{+} \bigcup v$
     - If flag = del,
       - If $v \in R_o$, set $R_{st}^{+} := R_{st}^{+} \bigcup v \,||\, \text{del} \,||\, \text{old}$
       - Else, set $R_{st}^{-} := R_{st}^{-} \bigcup v$

FIX

Push (Part 2)

Push $\left[K, \text{st}, \mu ; \text{EMM}\right]$:

2. Remove the tuple of $\ell_{\pi(j)}$ from MM_{st}

3. Set $\bar{v}_n := \left[\text{R}_n \bigcup \text{R}^{+}_{\text{st}}\right] \setminus \text{R}^{-}_{\text{st}}$

4. Compute

$\left[\perp ; \text{EMM'}\right] \leftarrow \text{DVLH.Delete}\left[K_o, \text{st}_o, \ell_{\pi(j)} ; \text{EMM}_o\right]$

$\left[\perp ; \text{EMM'}\right] \leftarrow \text{DVLH.Insert}\left[K_n, \text{st}_n, \ell_{\pi(j)}, \bar{v}_n ; \text{EMM}_n\right]$

4. Increment the counter cnt by μ

5. Client outputs the updated state st'

   Server outputs the updated multi-map EMM'

FIX

Leakage

- Setup leakage
  - The size of the input multi-map
  - The size of the label space
- Get leakage
  - The query equality pattern
- Inserts and Erases leakage
  - None

FIX

Efficiency when $\mu = O(\log N)$

- Get complexity $O(\log N)$
- Insert/Erase complexity $O(\log^3 N)$
- Storage complexity $O(\log N \cdot \#\mathbb{L}_{\mathsf{MM}})$
- Interactive
  - 2 rounds
- Stateful
- Lossy

FIX

Client Stash Analysis

Theorem 2. If the update labels are sampled uniformly at random and if their update lengths are sampled uniformly at random from U, then the expected stash size of FIX is at most $H_m \cdot s_{\max}/2\mu$ where

$$U = \left\{ \frac{s_0}{1^s \cdot H_{m,s}}, \frac{s_0}{2^s \cdot H_{m,s}}, \ldots, \frac{s_0}{m^s \cdot H_{m,s}} \right\}$$

Theorem 3. If the update labels sampled uniformly at random and if their tuple lengths are sampled from a Zipf $\mathcal{Z}_{m,1}$ distribution, then, with probability at least $1 - \epsilon$ the stash size is at most

$$\frac{H_m \cdot s_{\max}}{2\mu} + s_{\max} \cdot \sqrt{\frac{(m - 2\mu) \cdot \ln(1/\epsilon)}{2\mu}}$$

Thank you

https://eprint.iacr.org/2023/533