Exploiting Update Leakage In Searchable Symmetric Encryption

Jacob Haltiwanger and Thang Hoang
Virginia Tech, Department of Computer Science

This material is based upon work supported by the National Science Foundation under Grants Number 194649, an unrestricted gift from Robert Bosch, 4-VA, and the Commonwealth Cyber Initiative (CCI) — an investment in the advancement of cyber R&D, innovation, and workforce development. For more information about CCI, visit www.cyberinitiative.org.

VIRGINIA TECH™

Motivation

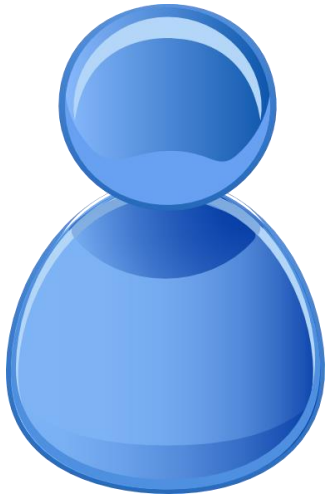
- Cloud computing offers popular and useful applications:
- Remote storage
- Scalable databases
- Email platforms

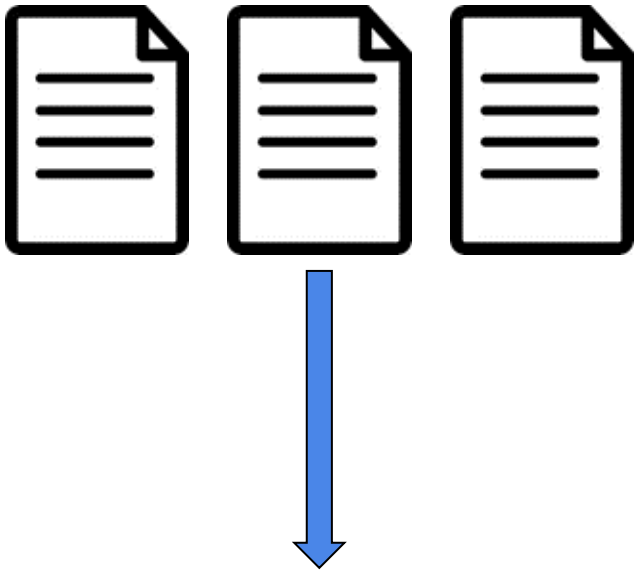
Motivation

How can a client search over private data on an untrusted remote server?

Searchable Symmetric Encryption: Keyword Extraction

Client

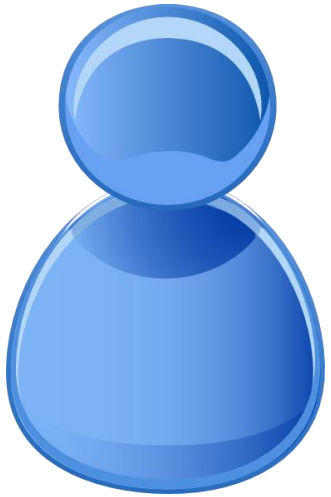




Apple: File 1, File 2, File 3
Banana: File 2
Carrot: File 9, File 22
…

Tokenizing the Index

Client



Inverted Index

Apple: File 1, File 2, File 3
Banana: File 2
Carrot: File 9, File 22
…

Apple
Banana
Carrot
…

One-Way Trapdoor

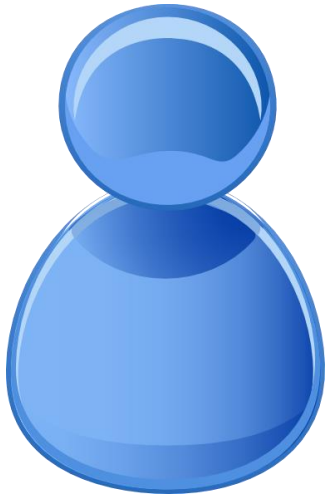
τ_1
τ_2
τ_3
…

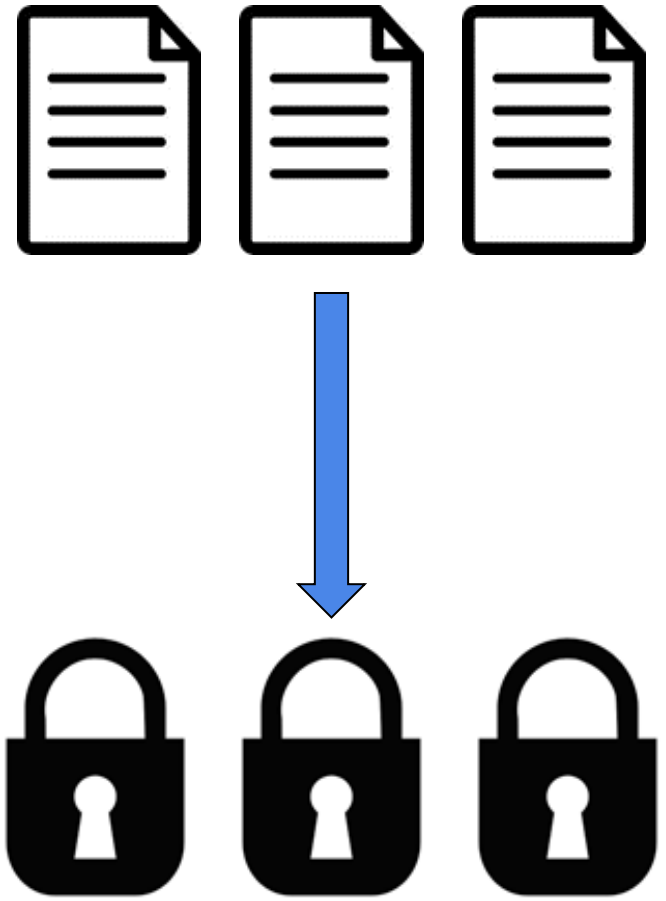
Encrypted Search Index

τ_1: File 1, File 2, File 3
τ_2: File 2
τ_3: File 9, File 22
…

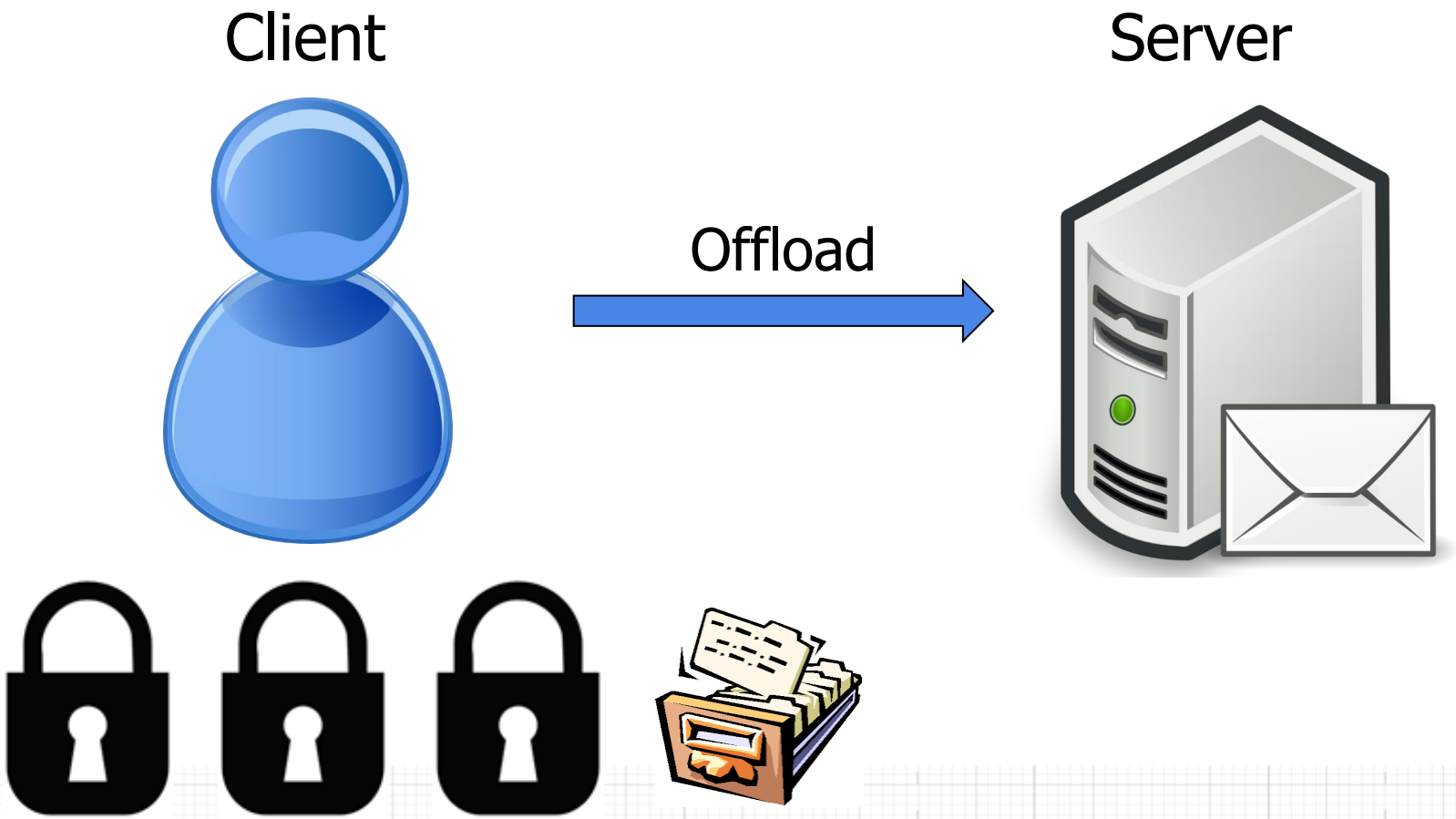
Encrypting the Files

Client



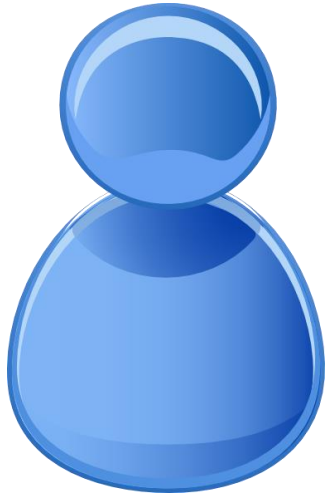


Completing the Setup Phase



Completing the Setup Phase

Client

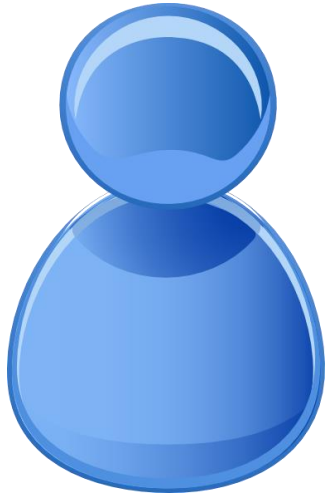


Server

τ_1: File 1, File 2, File 3
τ_2: File 2
τ_3: File 9, File 22
…

Searching the Encrypted Database

Client



One-Way Trapdoor

Banana → τ_2

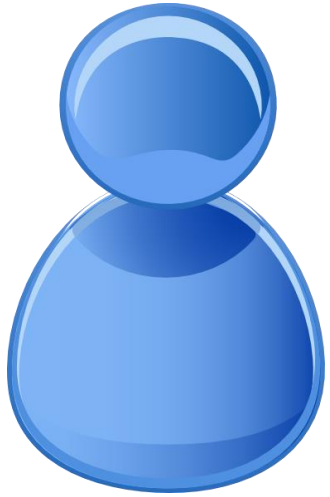
Server

τ_1: File 1, File 2, File 3
τ_2: File 2
τ_3: File 9, File 22
…

Searching the Encrypted Database

Client

Server



Client sends τ_2

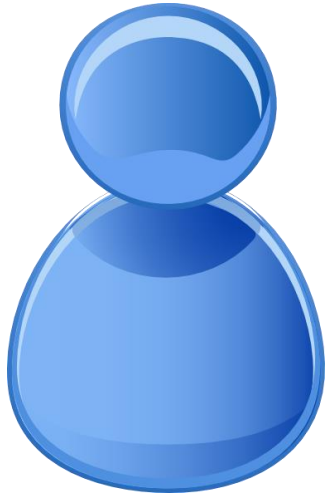
τ_1: File 1, File 2, File 3
τ_2: File 2
τ_3: File 9, File 22
…

One-Way Trapdoor

Banana  τ_2

Searching the Encrypted Database

Client

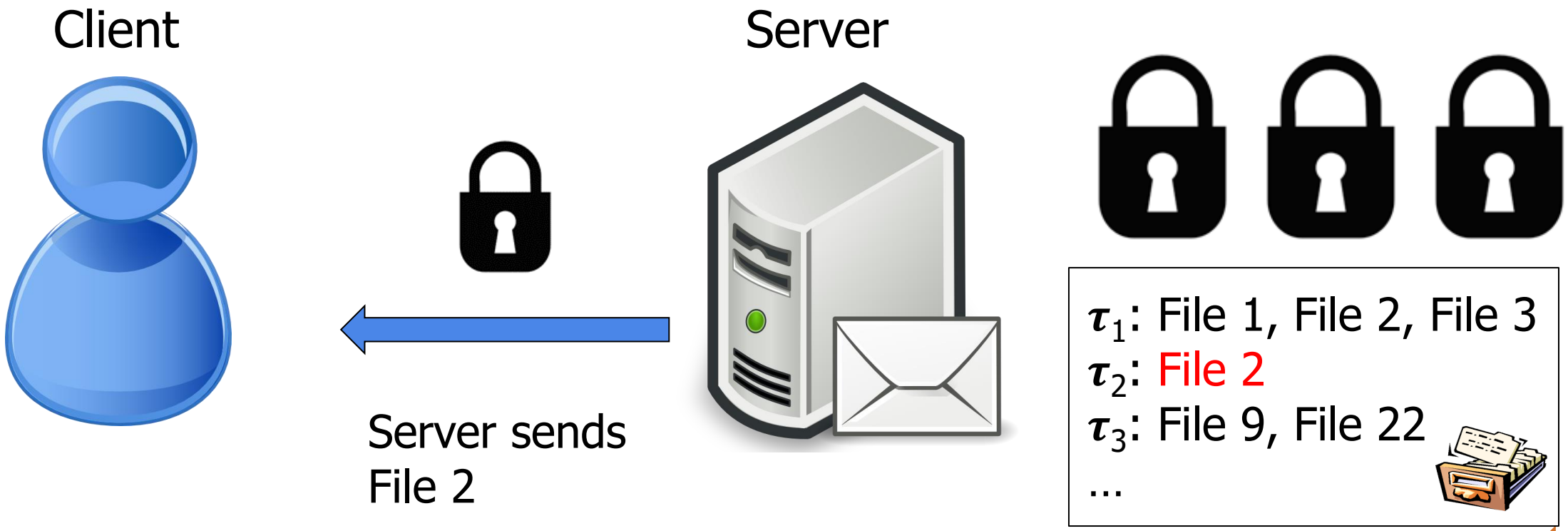


Client sends τ_2

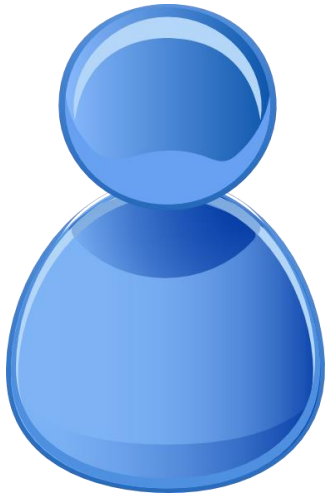
Server

τ_1: File 1, File 2, File 3
τ_2: File 2
τ_3: File 9, File 22
…

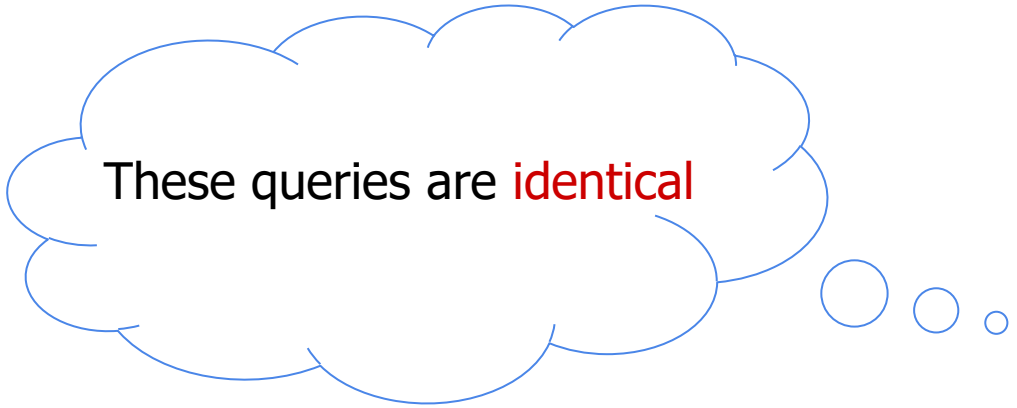
Searching the Encrypted Database



Vulnerability: Search Pattern Leakage



Client



Server

Keyword k → Query token t

One way function

This process is deterministic

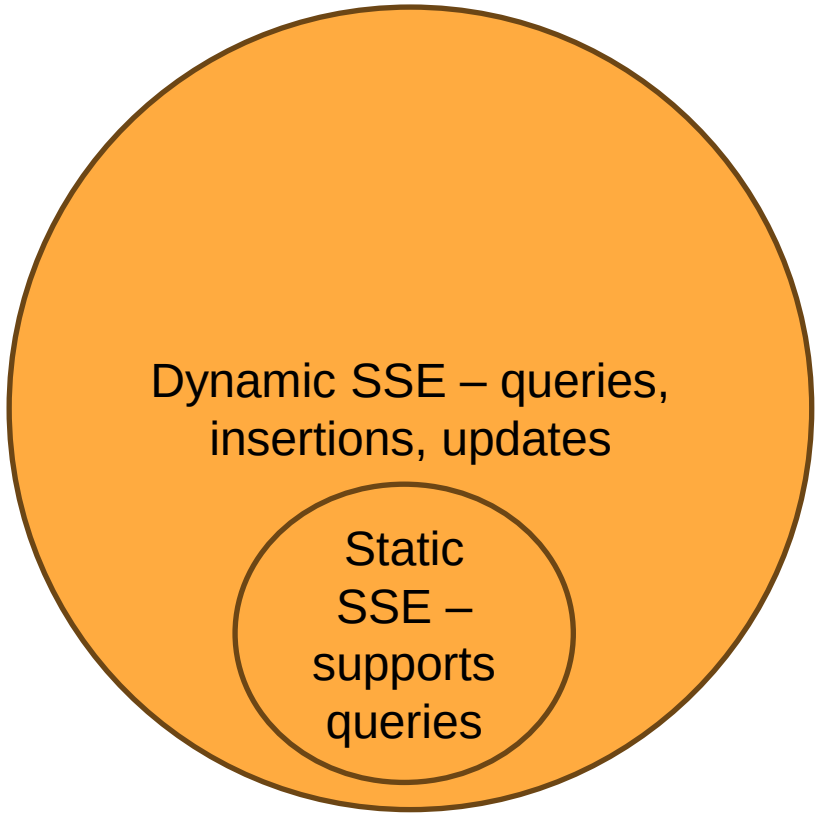
$\{\tau_1, \tau_2, \tau_3, \tau_4, \tau_2, \tau_1, \tau_5\}$

Example list of previous queries

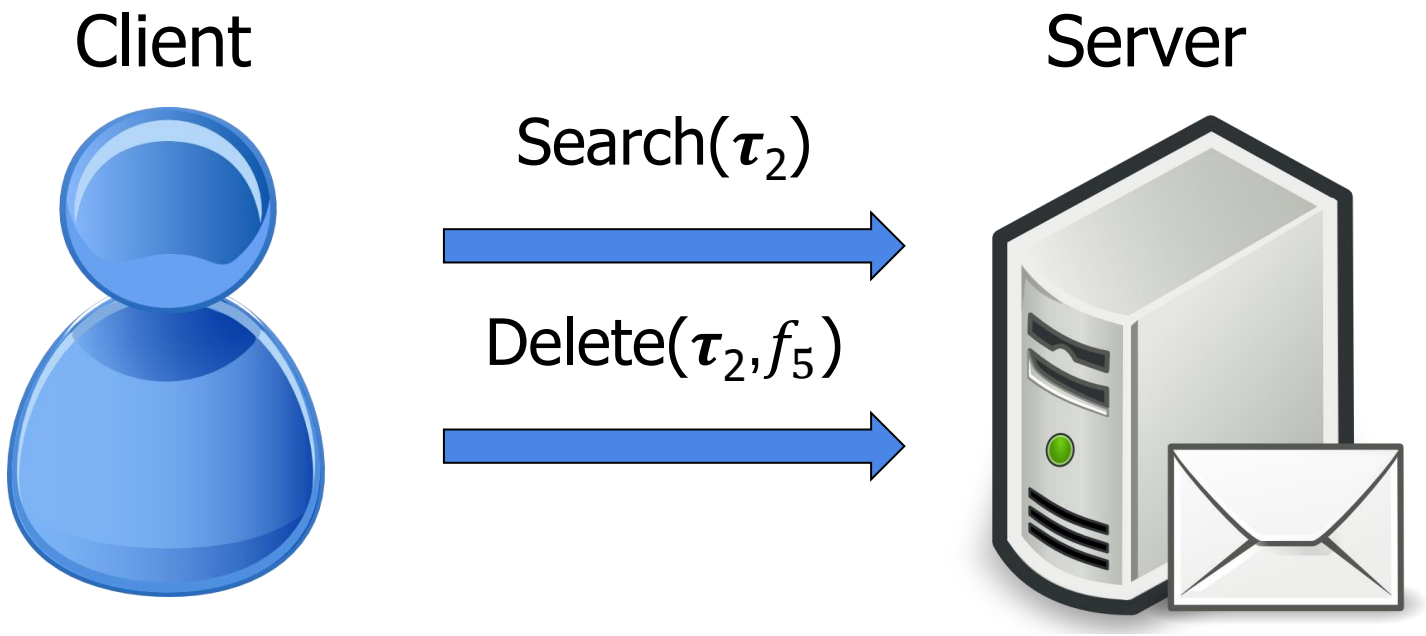
Dynamic SSE

- We focus on **Dynamic** SSE, which enables index updates, ex.
- Insert($\boldsymbol{\tau}_2$, f_5), Delete($\boldsymbol{\tau}_4$, f_1)

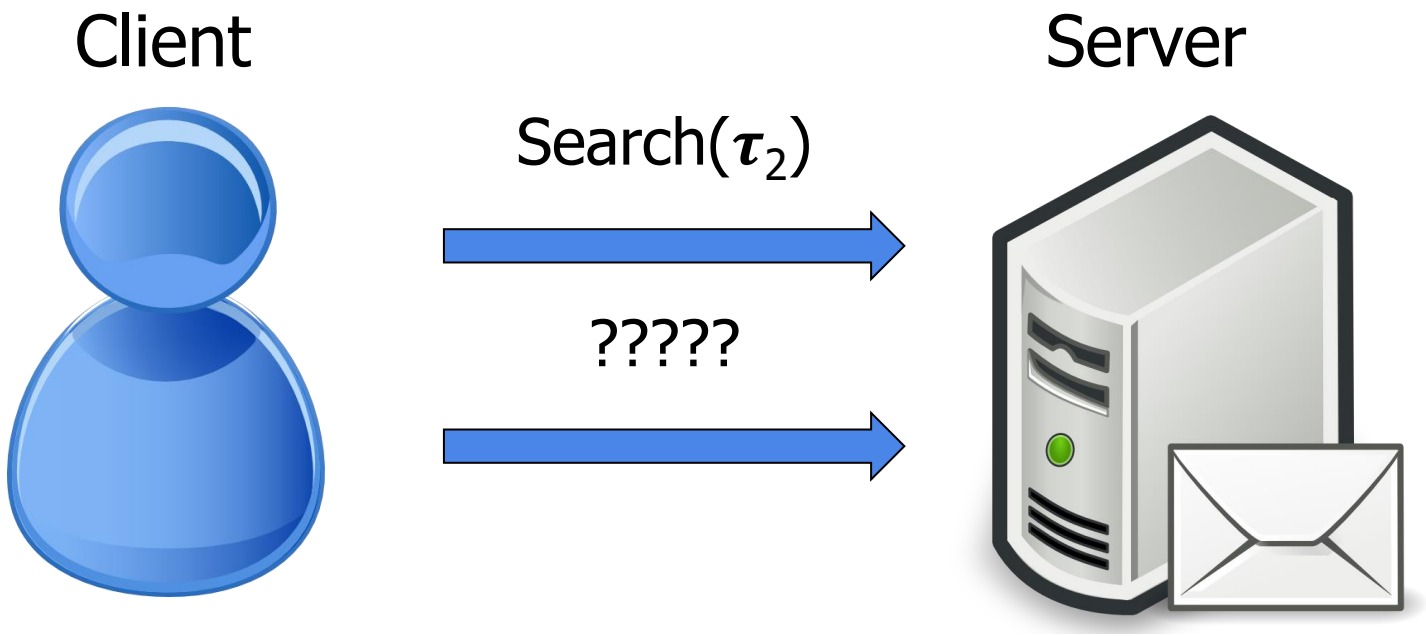
- Forward and Backward Privacy are optional security properties



Forward Privacy



Forward Privacy



Server cannot link future updates to previous queries

Insight

- We can learn info about updates after a search
- For example…

Post-Search Update Leakage

- Assume a client does the following operations:

1. Search($\boldsymbol{\tau}_1$)

The server sees all boxed operations.

Post-Search Update Leakage

- Assume a client does the following operations:

1. Search($\boldsymbol{\tau}_1$)
2. Add($\boldsymbol{\tau}_1, f_1$)

The server sees all boxed operations.

Post-Search Update Leakage

- Assume a client does the following operations:

1. Search($\boldsymbol{\tau}_1$)
2. Add($\boldsymbol{\tau}_1, f_1$)
3. Delete($\boldsymbol{\tau}_2, f_1$)

The server sees all boxed operations.

Post-Search Update Leakage

- Assume a client does the following operations:

1. Search($\boldsymbol{\tau}_1$)
2. Add($\boldsymbol{\tau}_1, f_1$)
3. Delete($\boldsymbol{\tau}_2, f_1$)
4. Search($\boldsymbol{\tau}_1$)

The server sees all boxed operations.

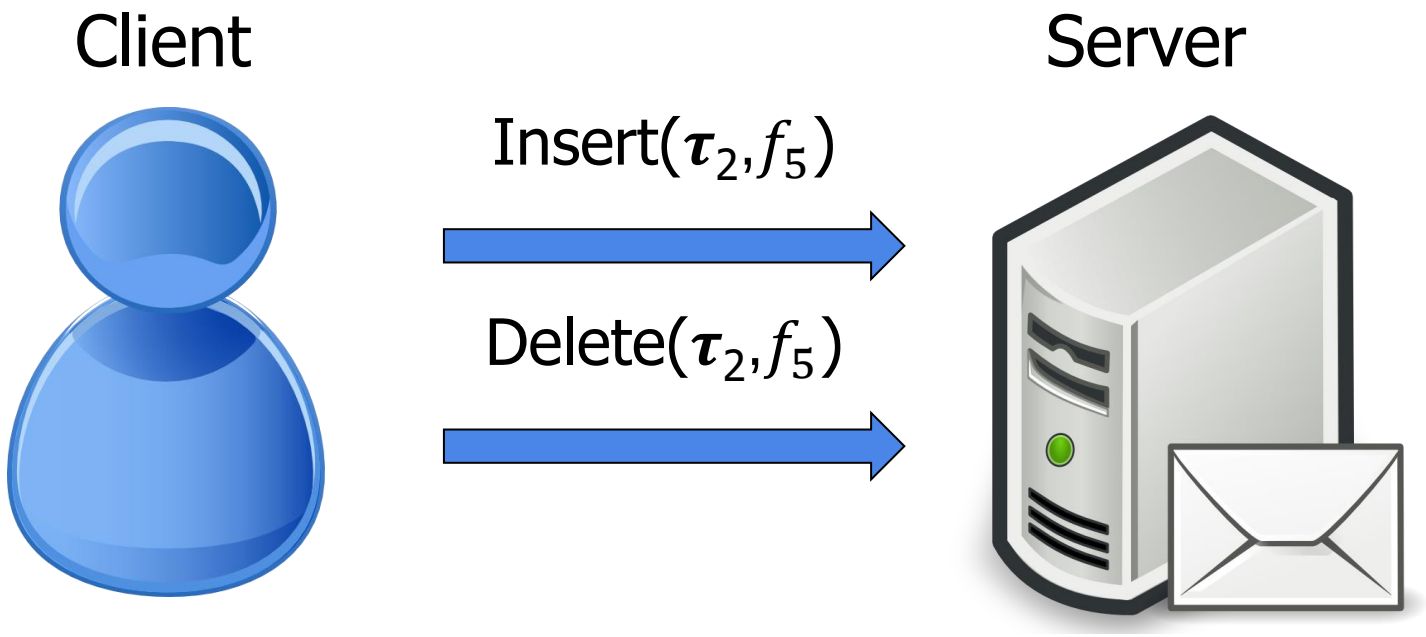
Post-Search Update Leakage

- Assume a client does the following operations:

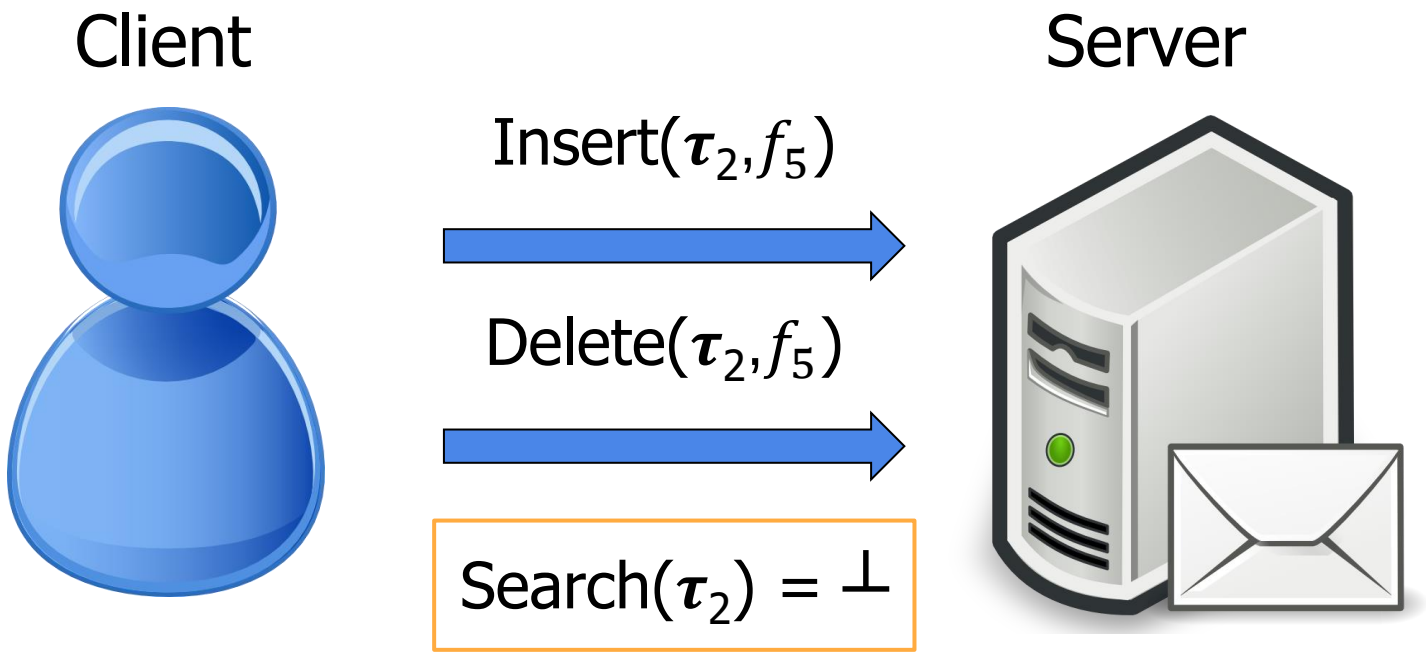
1. Search($\boldsymbol{\tau}_1$)
2. Add($\boldsymbol{\tau}_1, f_1$)
3. Delete($\boldsymbol{\tau}_2, f_1$)
4. Search($\boldsymbol{\tau}_1$)
5. Search($\boldsymbol{\tau}_2$)

The server sees all boxed operations.

Backward Privacy



Backward Privacy



Server cannot learn about contents of deleted entries

Backward Privacy

- After client searches for τ, they learn the following about previous updates on τ:
- **No BP: Full update history (operation, file identifiers, timestamps of all updates)**
- Level 3 BP: Which deletions cancelled which additions, timestamps of all updates
- **Level 2 BP: Timestamps of all updates**
- Level 1 BP: Total number of updates

Backward Privacy

- After client searches for τ, they learn the following about previous updates on τ:
- **No BP: Full update history (operation, file identifiers, timestamps of all updates)**
- Level 3 BP: Which deletions cancelled which additions, timestamps of all updates
- **Level 2 BP: Timestamps of all updates**
- Level 1 BP: Total number of updates

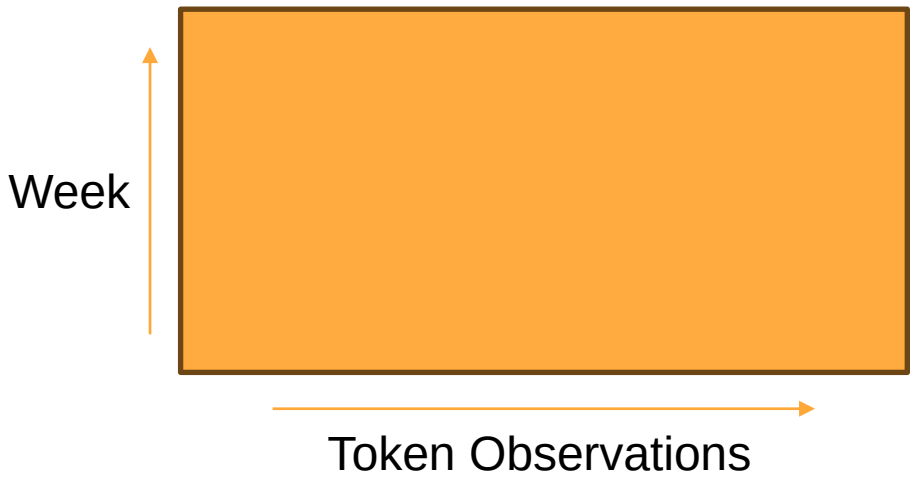
Consensus has been that less leakage must be better, but…
BP reduces performance, so a concrete understanding of the tradeoff is desirable

Insights

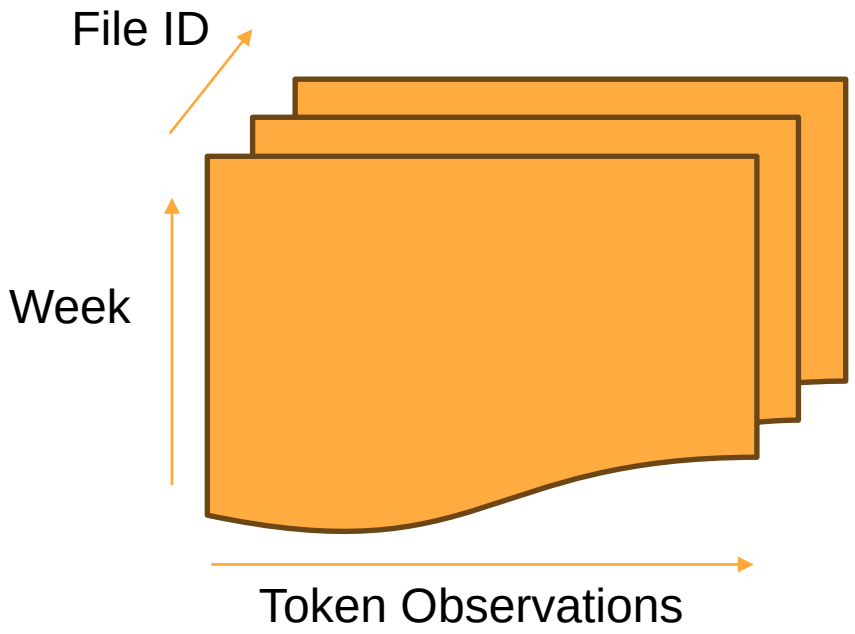
- Level 2 BP and worse: we can use update **timestamps** for a frequency attack
- No BP: we can also exploit the **file identifier**

Approach

UF Attack
(Update Frequency)



UFID Attack
(Update Frequency with File IDs)



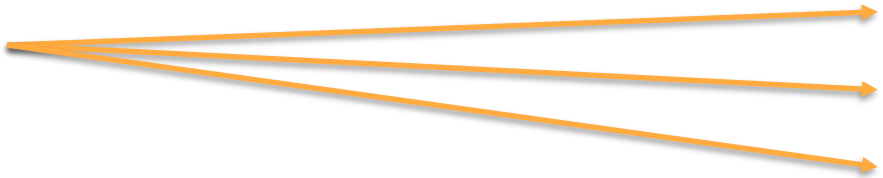
Cross-reference with auxiliary info!

<u>Auxiliary Info</u>

Apple, 7 updates, f_3

Banana, 18 updates, f_1

Coconut, 4 updates, f_3

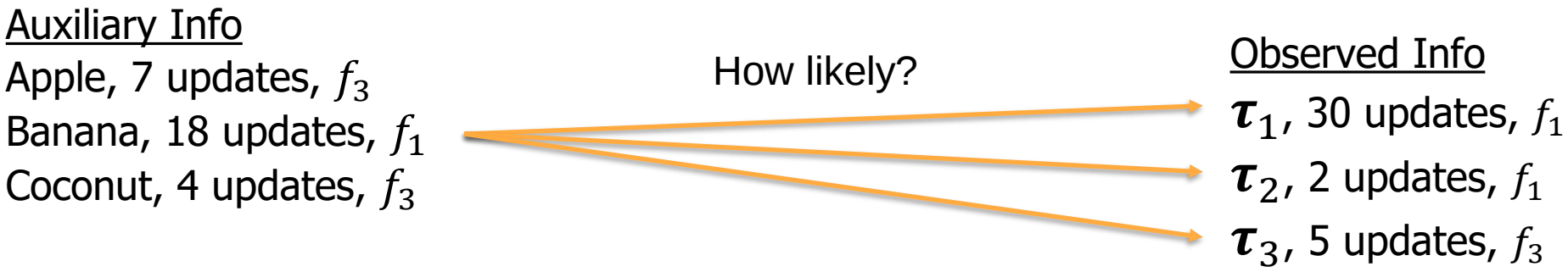


<u>Observed Info</u>

$\boldsymbol{\tau}_1$, 30 updates, f_1

$\boldsymbol{\tau}_2$, 2 updates, f_1

$\boldsymbol{\tau}_3$, 5 updates, f_3



Optimize to find most likely mapping

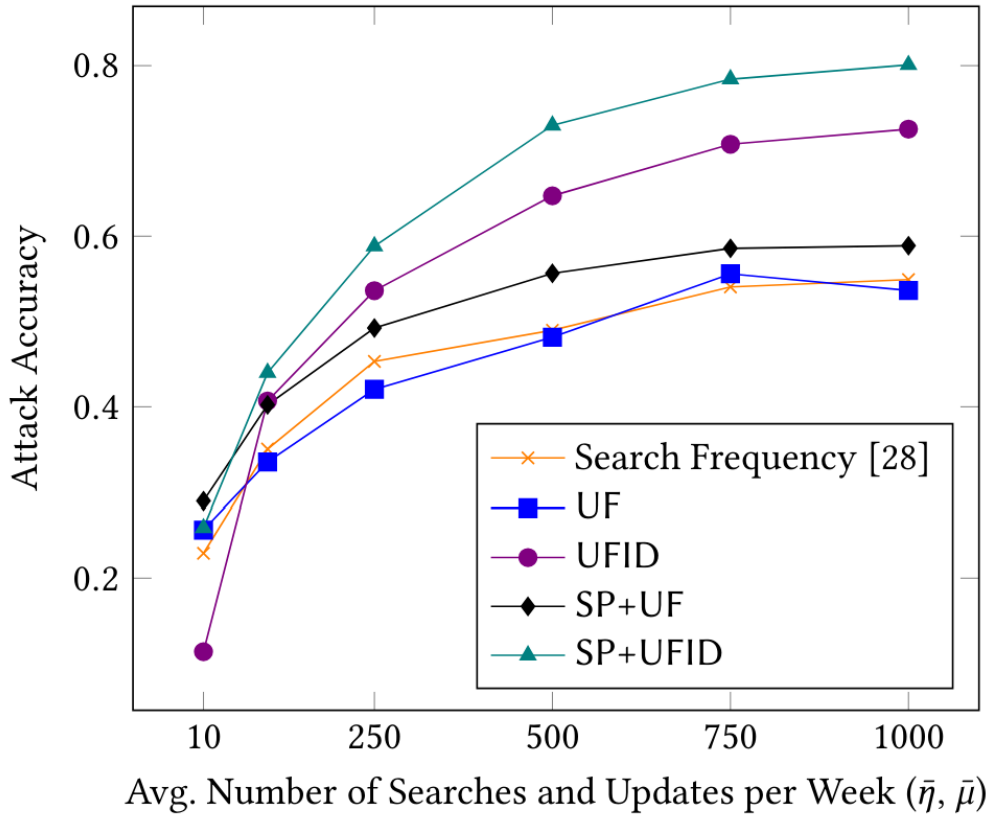
Approach

- Element-by-element multiplication gives joint probability of events occurring
- Can combine with Oya and Kerschbaum's Search Frequency attack to make full use of the search patterns
  - SP+UF Attack (Level II BP)
  - SP+UFID Attack (No BP)

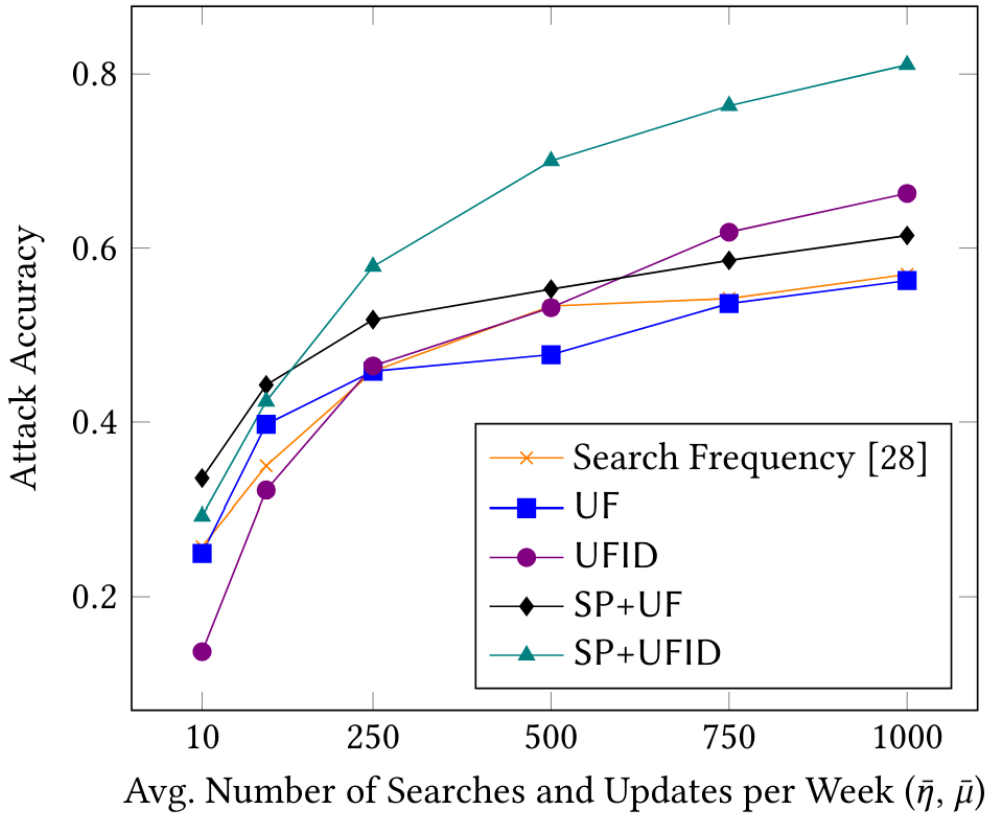
Evaluation

- Extracted keywords from Enron and Lucene datasets
- 100 weeks; first 50 are auxiliary data, adversary tries to guess last 50
- Model searches and updates based on Google Trends probabilities
- Apply forward privacy: updates only revealed if they are searched later
- Apply backward privacy for UF/SP+UF Attacks: hide file ID from update tuples
- Accuracy is number of keywords correctly guessed divided by total keywords seen

No Backward Privacy is a Problem…

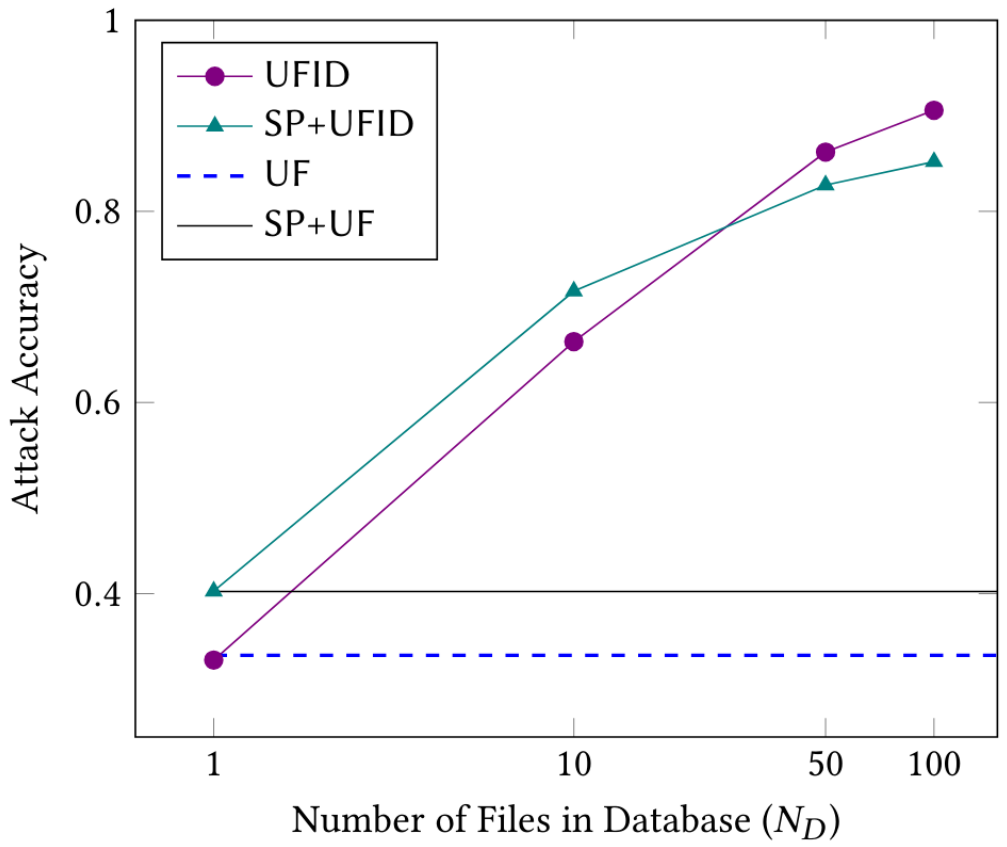


(a) Enron dataset

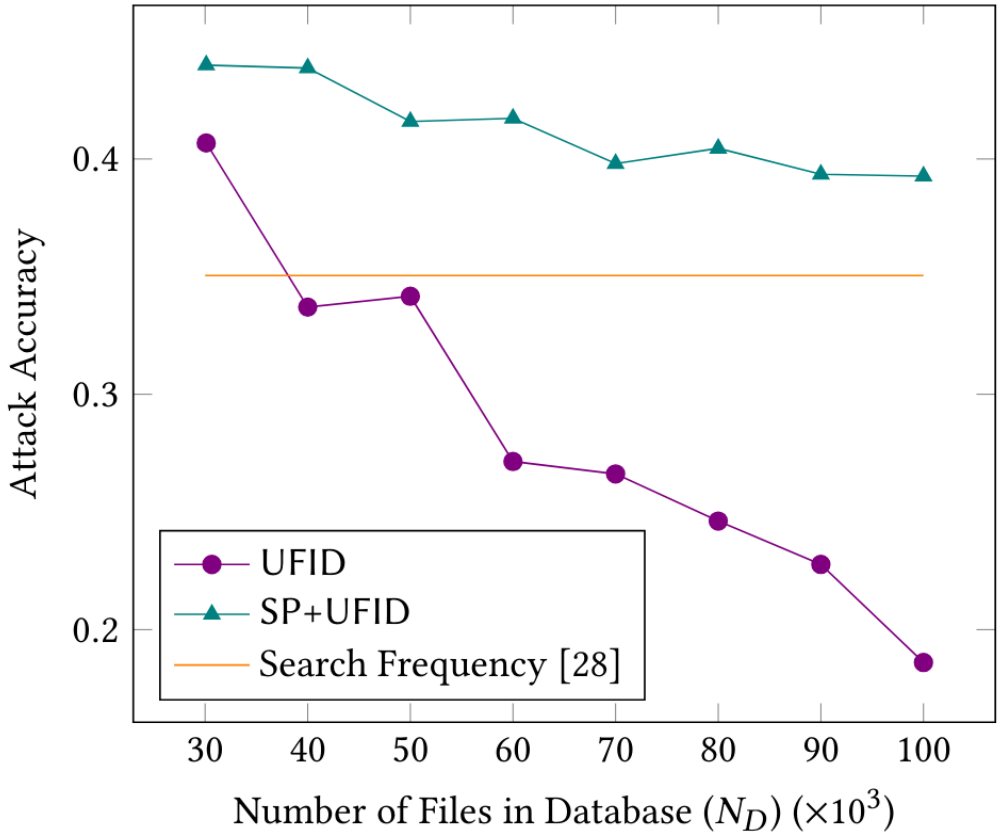


(b) Lucene dataset

Combining Leakage is the Way to Go



(a) Small database



(b) Large database

Limitations

- Obtaining auxiliary info is hard!
- Tracking search patterns in DSSE is non-trivial
- Doesn’t exploit update operation type (add or del)
- Defeated by Level 1 BP or search pattern hiding

Contributions and Conclusion

- We present the first attacks to exploit Post-Search Update Leakage in Forward and Backward-Private DSSE schemes
- Our attacks are the first to empirically validate that BP can reduce attack accuracy by a significant margin
- Improving efficiency of BP is important so users of commercial DSSE applications can enjoy better privacy

Questions?