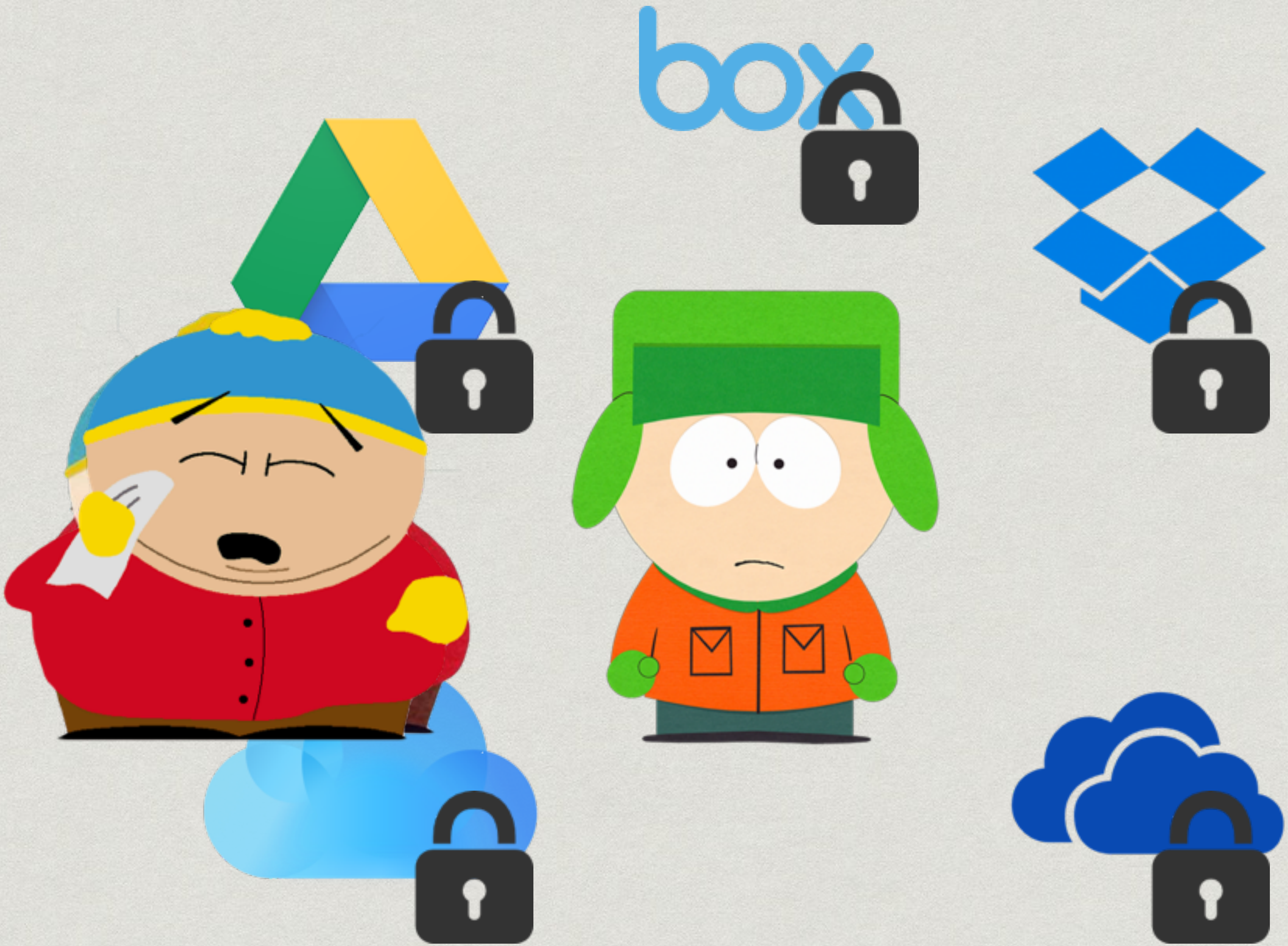
FORWARD PRIVATE SEARCHABLE ENCRYPTION

DATE 13/07/2016 🎂

MSR CAMBRIDGE - RAPHAEL BOST


box

Searchable Encryption

- Outsource data …
- … securely
- … keep search functionalities

Generic Solutions

We can use generic tools to solve this problem:

- Fully Homomorphic encryption
  - Run all computations on the server
    Complexity linear in the DB size
- Oblivious RAM
  - Hide access pattern but…
    ORAM lower bound (logarithmic)

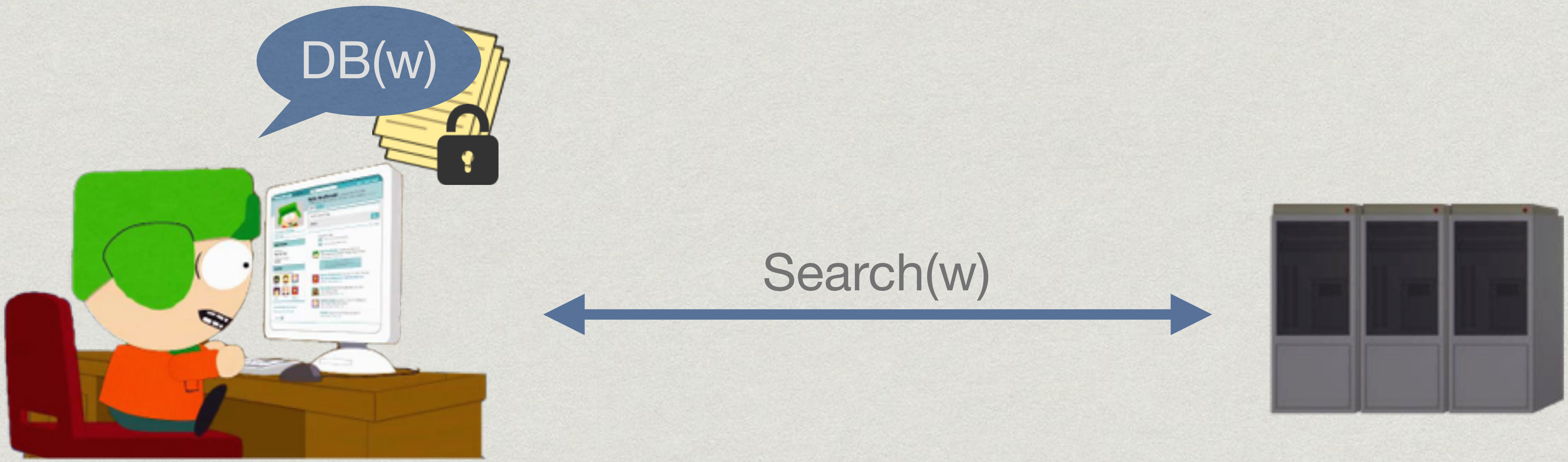
Ad-hoc Constructions

Can we get more efficient solutions?

- Yes, but …
- … we have to leak some information

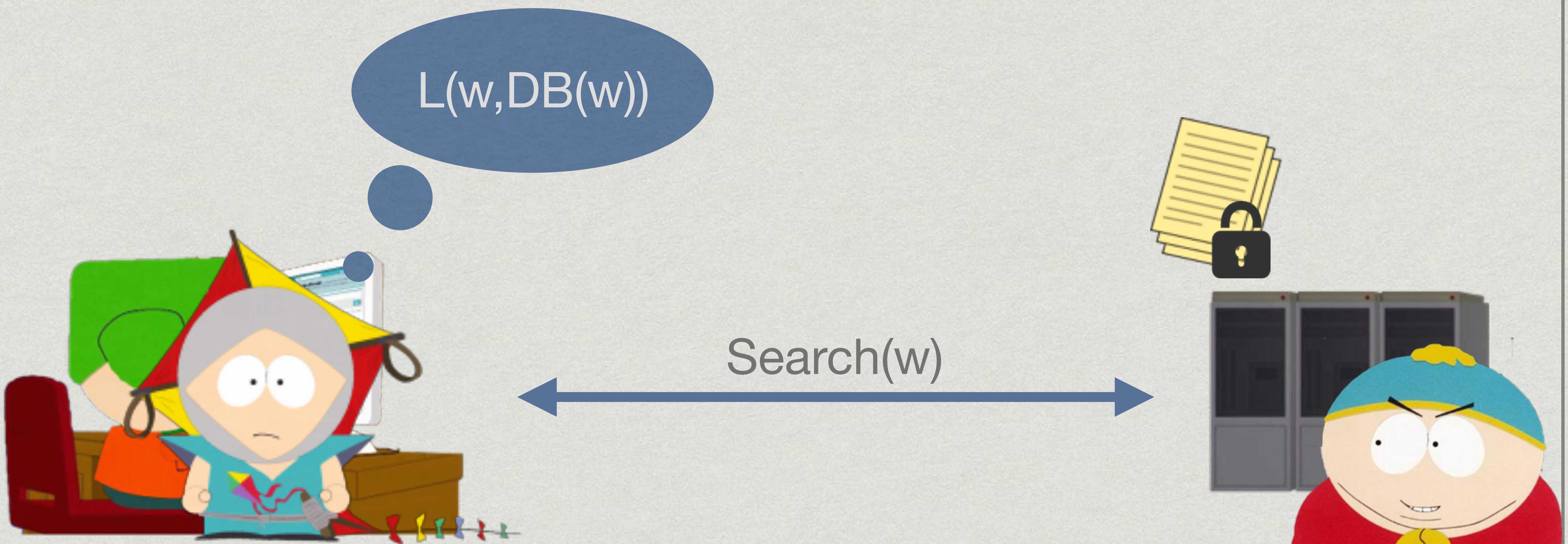
Security/performance tradeoff

Security of SE



Security of SE

Security of SE



Everything the server learns, he can compute from the leakage

The search protocol can be simulated from the leakage

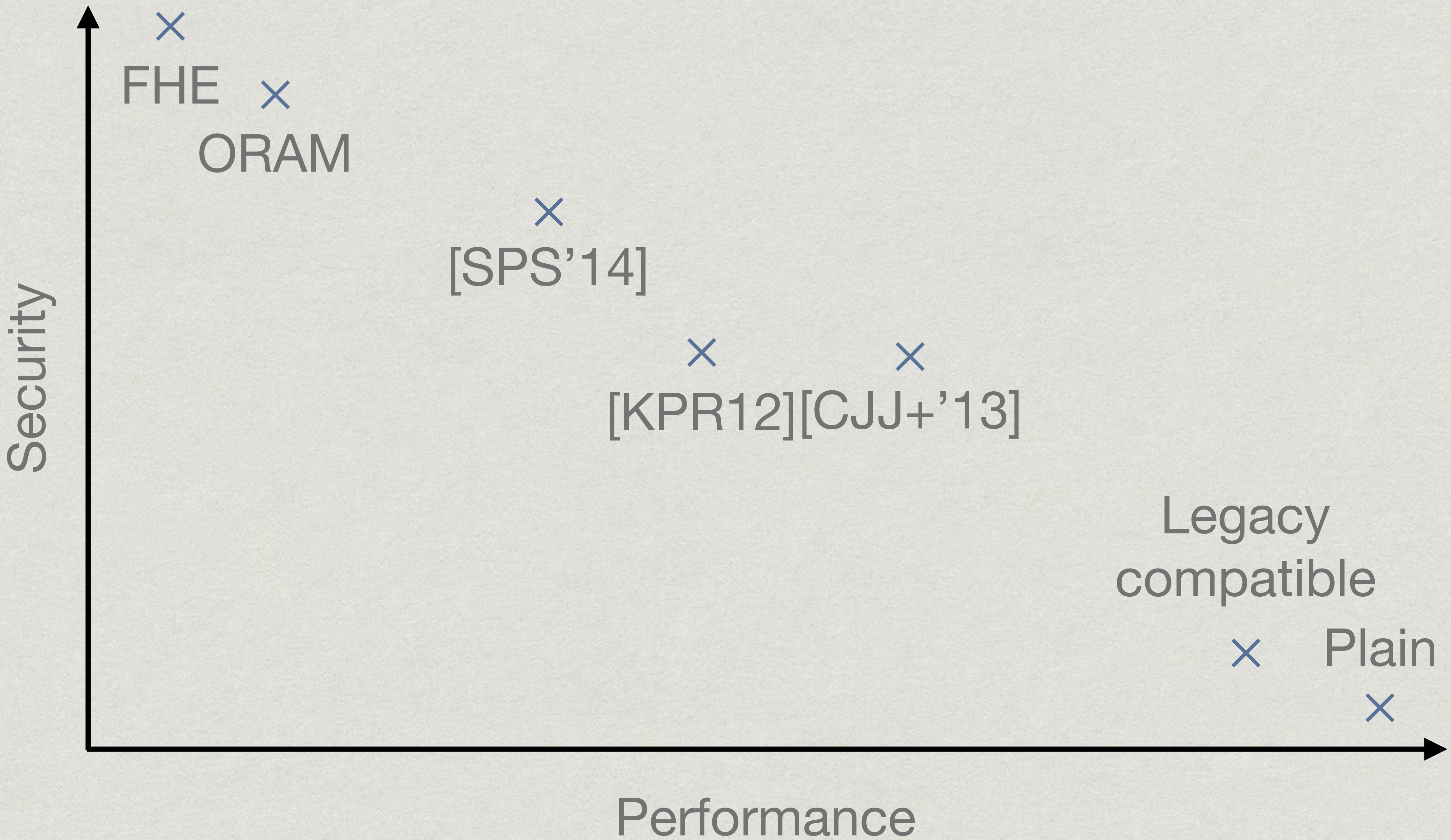
Common Leakage

* Search leakage :
  * repetition of queries (aka. search pattern)
  * results
* Update leakage:
  * updated documents
  * repetition of updated keywords
  * ...

Previous Results

- First constructions [SWP00]
- Formalization of the security model [CGKO06]
- Efficient dynamic constructions [KPR12]
- Boolean queries & scalability [CJJKRS13]
  - ↳ various extensions (dynamisms, wildcards, range queries, …)
- Reduced update leakage [SPS14]
- …

Security-Performance Tradeoff



Leakage-Abuse Attacks

- ‘Everything the server learns, he can compute from the leakage’
    - ➡ What can be computed from the leakage?
- Recover the queried keywords from the leakage

‘Passive’ Attacks

- [IKK’12]: Using a co-occurrence probability matrix, the attacker can recover from 100% to 65% of the queries
- [CGPR’15]: Improvement of the IKK attack, 100% recovery
    - ➡ Use padding as a countermeasure

‘Active’ Attacks

- [ZKP’16]: Non-adaptive file injection attacks
  - Insert purposely crafted documents in the DB. Use binary search to recover the query

D_1	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8
D_2	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8
D_3	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8

log K injected documents

‘Active’ Attacks

- [ZKP’16]: Non-adaptive file injection attacks
  - Insert purposely crafted documents in the DB. Use binary search to recover the query
  - Counter measure: no more than T kw./doc.

(K/T) · log T injected documents

  - Adaptive version of the attack

(K/T) + log T injected documents

‘Active’ Adaptive Attacks

- [ZKP’16]: File injection attacks
  - Adaptive version of the attack

(K/T) + log T injected documents

  - If the attacker has prior knowledge about the database (e.g. frequency distribution)

log T injected documents

'Active' Adaptive Attacks

- All these adaptive attacks use the update leakage:
  - For an update, most SE schemes leak if the inserted document matches a previous query
  - We need SE schemes with oblivious updates

Forward Privacy

Forward Privacy

- An SE scheme is forward private if its update protocol does not leak any information about the updated keywords

$$L(op,w,ind) = L'(op, ind)$$

- Important feature: secure online build of the EDB

- Only one existing scheme so far [SPS'14]

  ➡ Very close to ORAM (logarithmic updates)

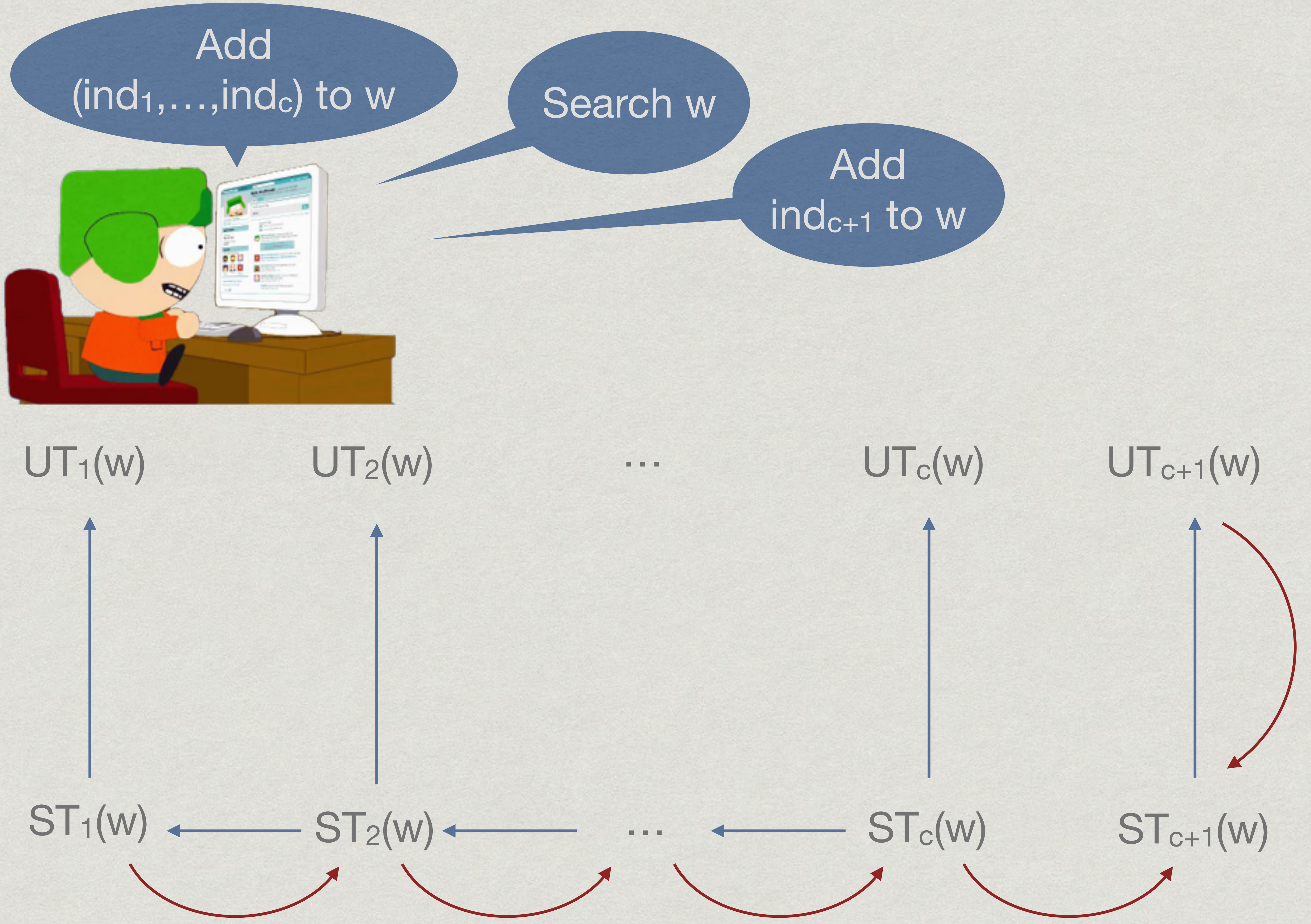
Add
$(ind_1, \ldots, ind_c)$ to w

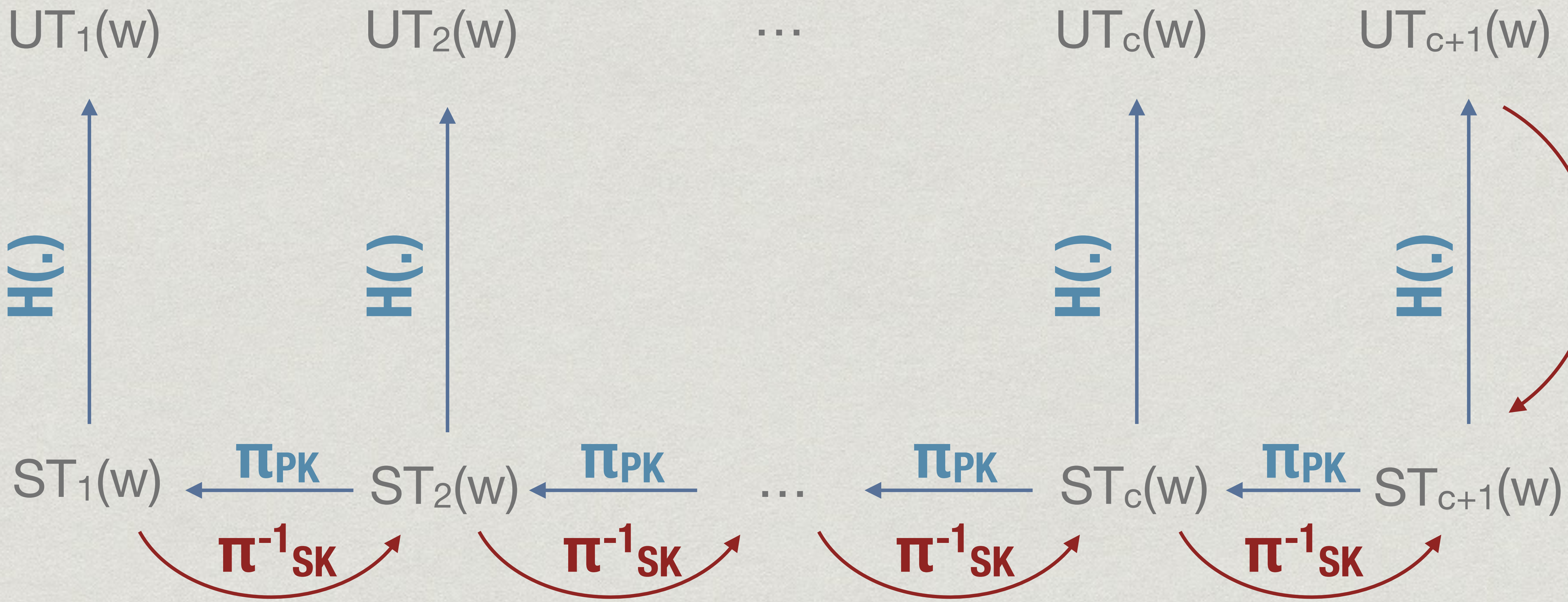
Search w

$UT_1(w)$ $UT_2(w)$ … $UT_c(w)$

…

$ST(w)$


Add
(ind$_1$,…,ind$_c$) to w
Search w
Add
ind$_{c+1}$ to w
UT$_1$(w)
UT$_2$(w)
…
UT$_c$(w)
UT$_{c+1}$(w)
ST$_1$(w)
ST$_2$(w)
…
ST$_c$(w)
ST$_{c+1}$(w)



- Naïve solution: $ST_i(w) = F(K_w, i)$
  - ✗ Client needs to send c tokens
  - ✗ Sending only K_w is not forward private
- Use a trapdoor permutation

Σοφος - Complexity

- Search(w): **Optimal**
  Client: O(1)
  Server: O(|DB(w)|)
- Update(+,w, ind): **Optimal**
  Client: O(1)
  Server: O(1)
- Storage:
  Client: O(K)
  Server: O(N) **Optimal**

Σοφος

- TDP π? RSA or Rabin
    - ✗ Elements (STs) are large (2048 bits).
    - ✗ Client storage is impractical
- Pseudo-randomly generate $ST_0(w)$, and compute $ST_c(w)$ on the fly (only c is stored by the client)
    - ✓ Efficient (non-iterative) using RSA
- Search is embarrassingly parallelizable

$$x^{d^{\cdot^{\cdot^{\cdot^{d}}}}} = x^{(d^c \bmod \phi(N))} \bmod N$$

Σοφος - Security

- Update leakage: nothing

Forward private

- Search leakage:
  - search pattern
  - ‘history’ of w: the timestamped list of updates of keyword w

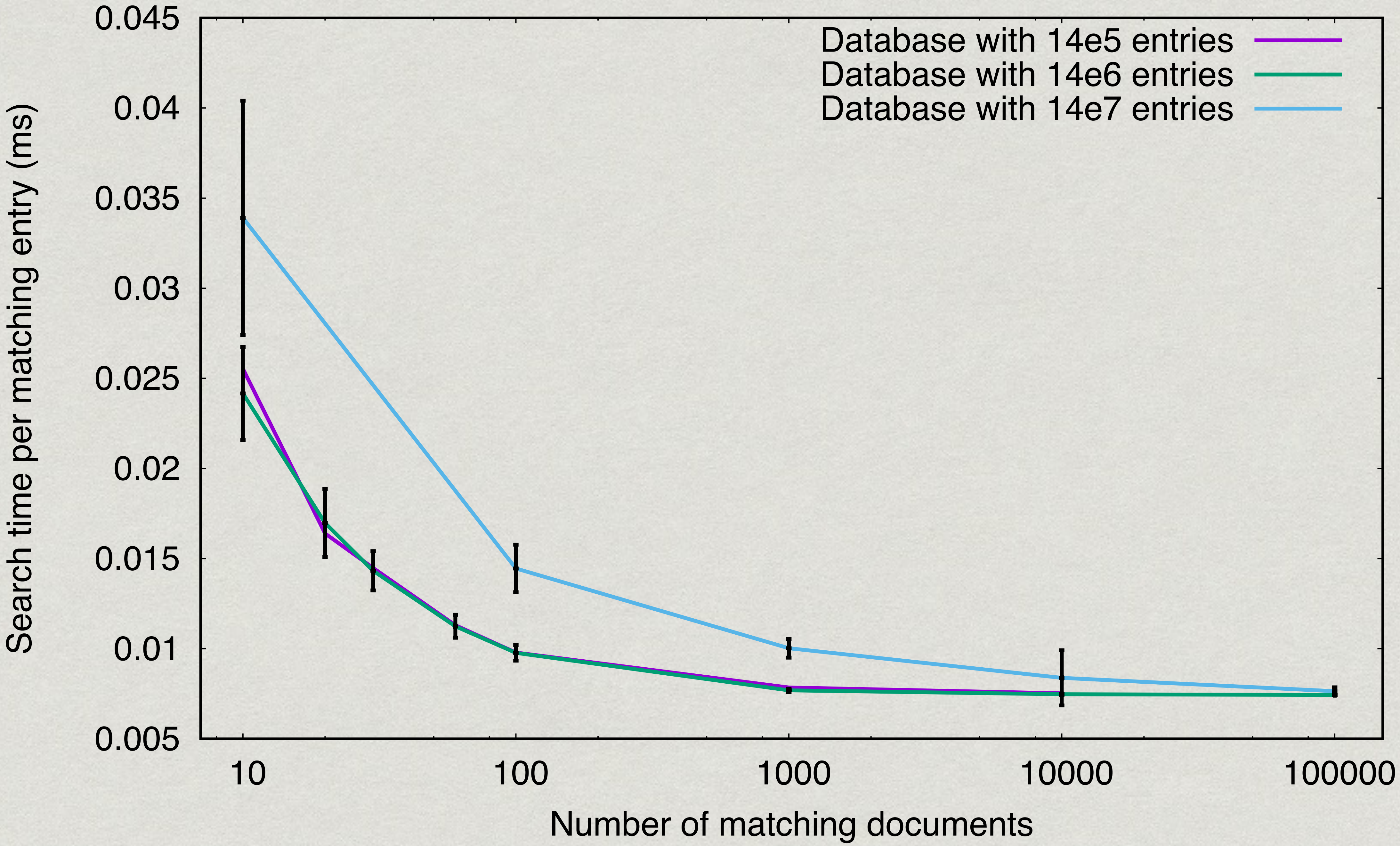
Adaptive security (ROM)

Σοφος - Evaluation

- C/C++ full fledged implementation
- Server KVS: RockDB
- Evaluated on a desktop computer
  4 GHz Core i7 CPU, 8GB RAM, SSD

https://gitlab.com/sse/sophos

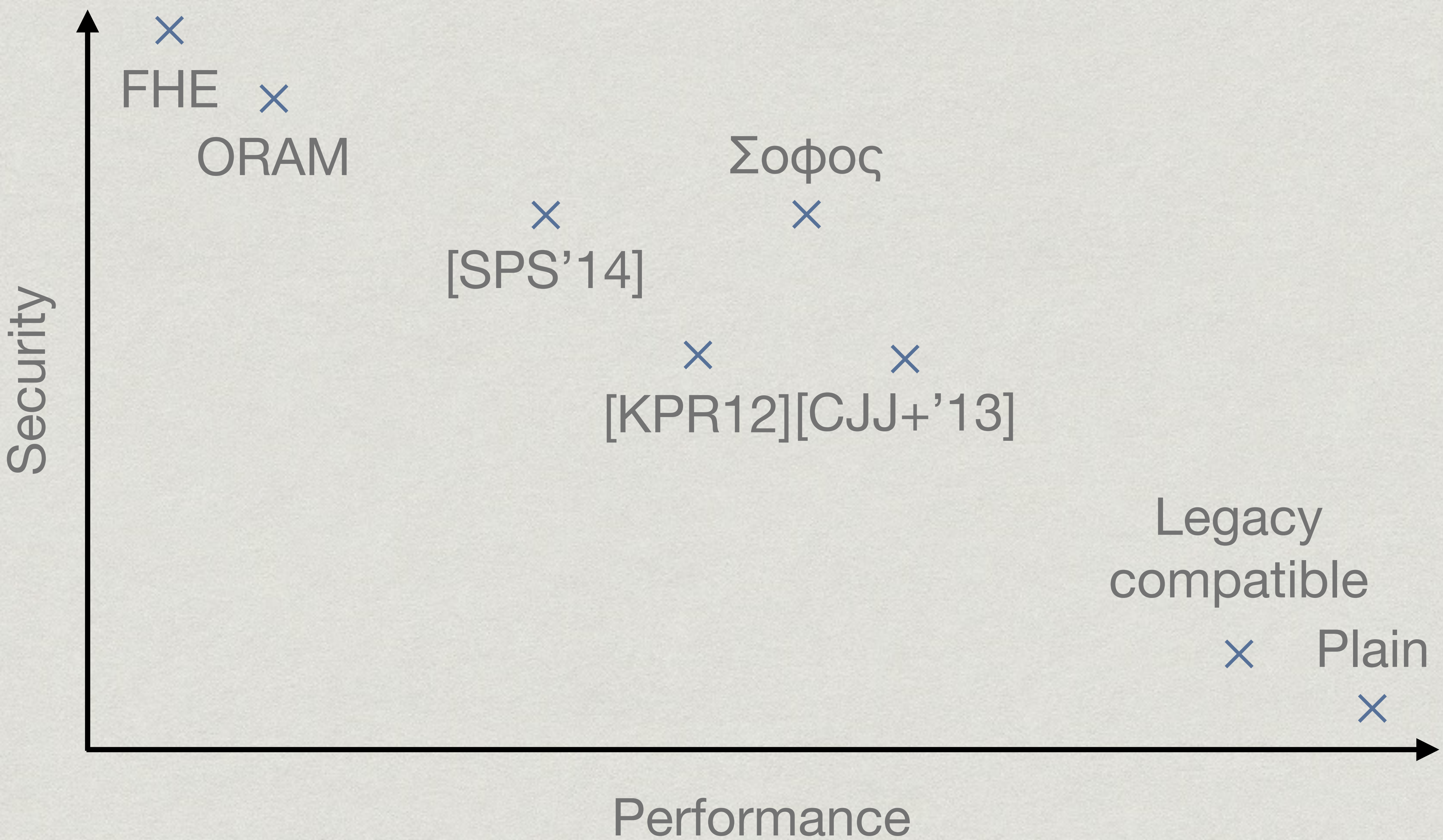
Σοφος - Evaluation



Σοφος

* Provable forward privacy
* Efficient search
* Asymptotically efficient update (optimal)
* In practice, very low update throughput (4300 p/s - 20x slower than other work)

Security-Performance Tradeoff



Ongoing/future work

* Improve the update throughput (get rid of RSA)
* Dynamic padding
* Thwart (non-adaptive) file injection attacks

THANKS!