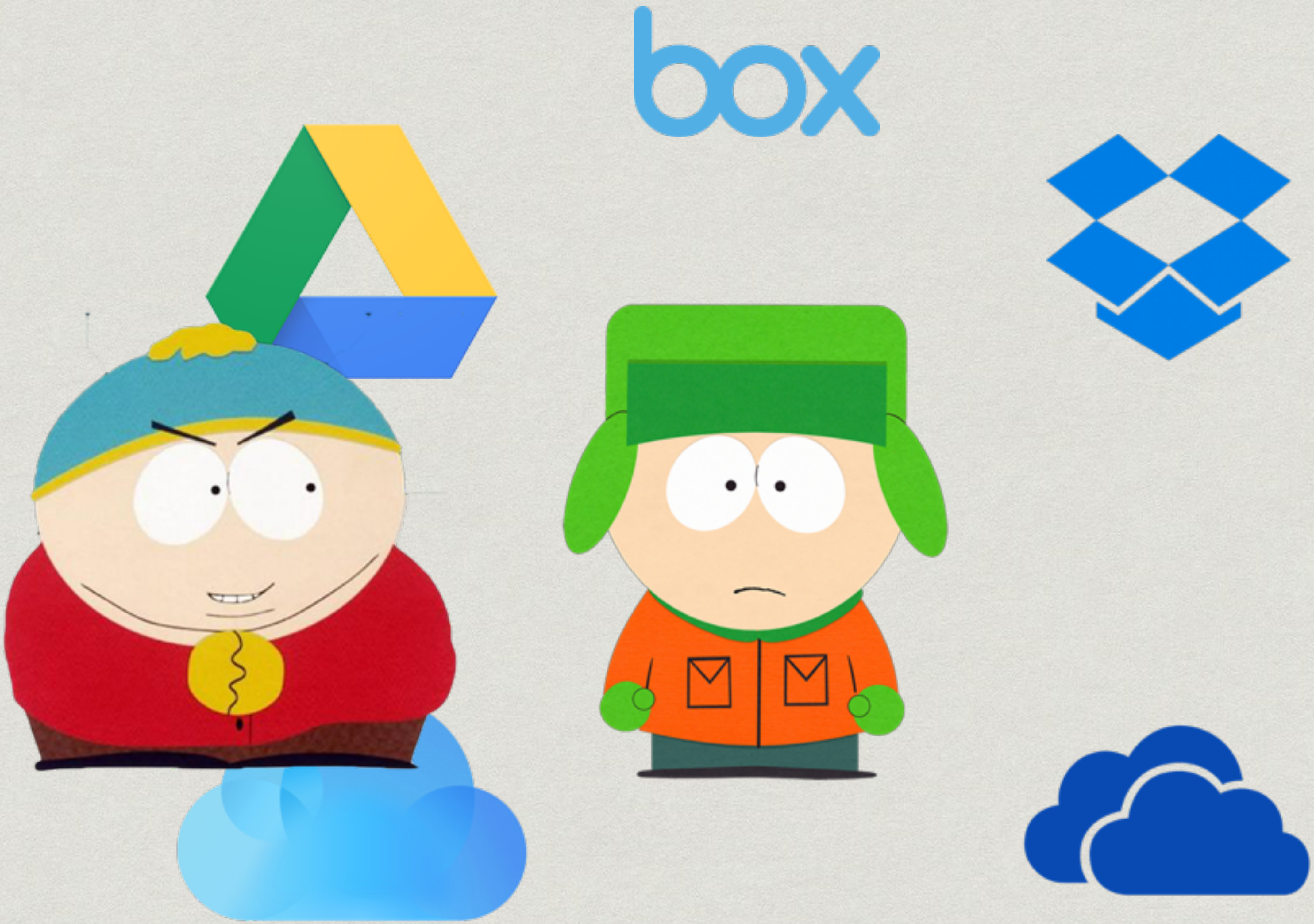
VERIFIABLE SYMMETRIC SEARCHABLE ENCRYPTION

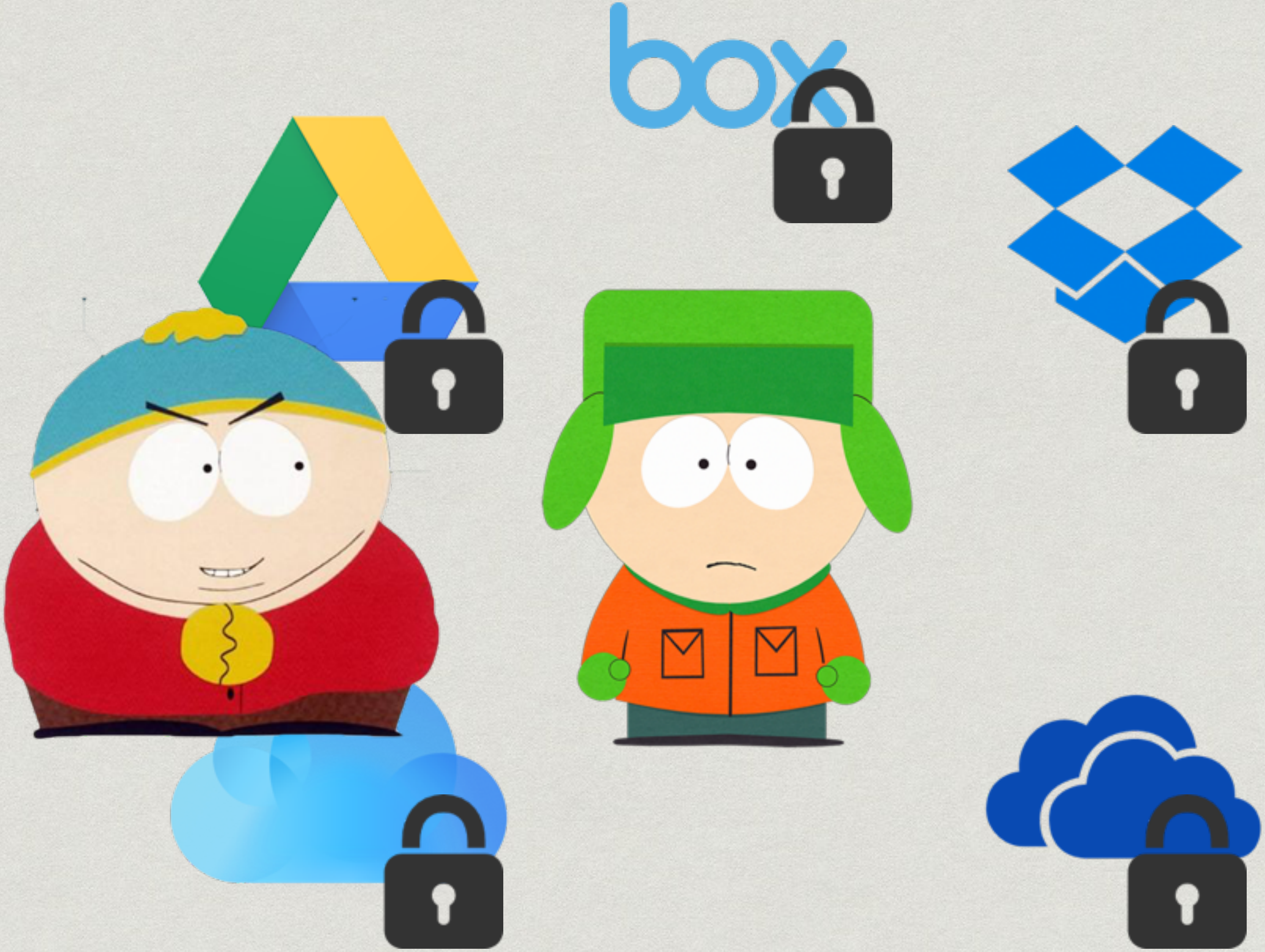
DATE 09/03/2016

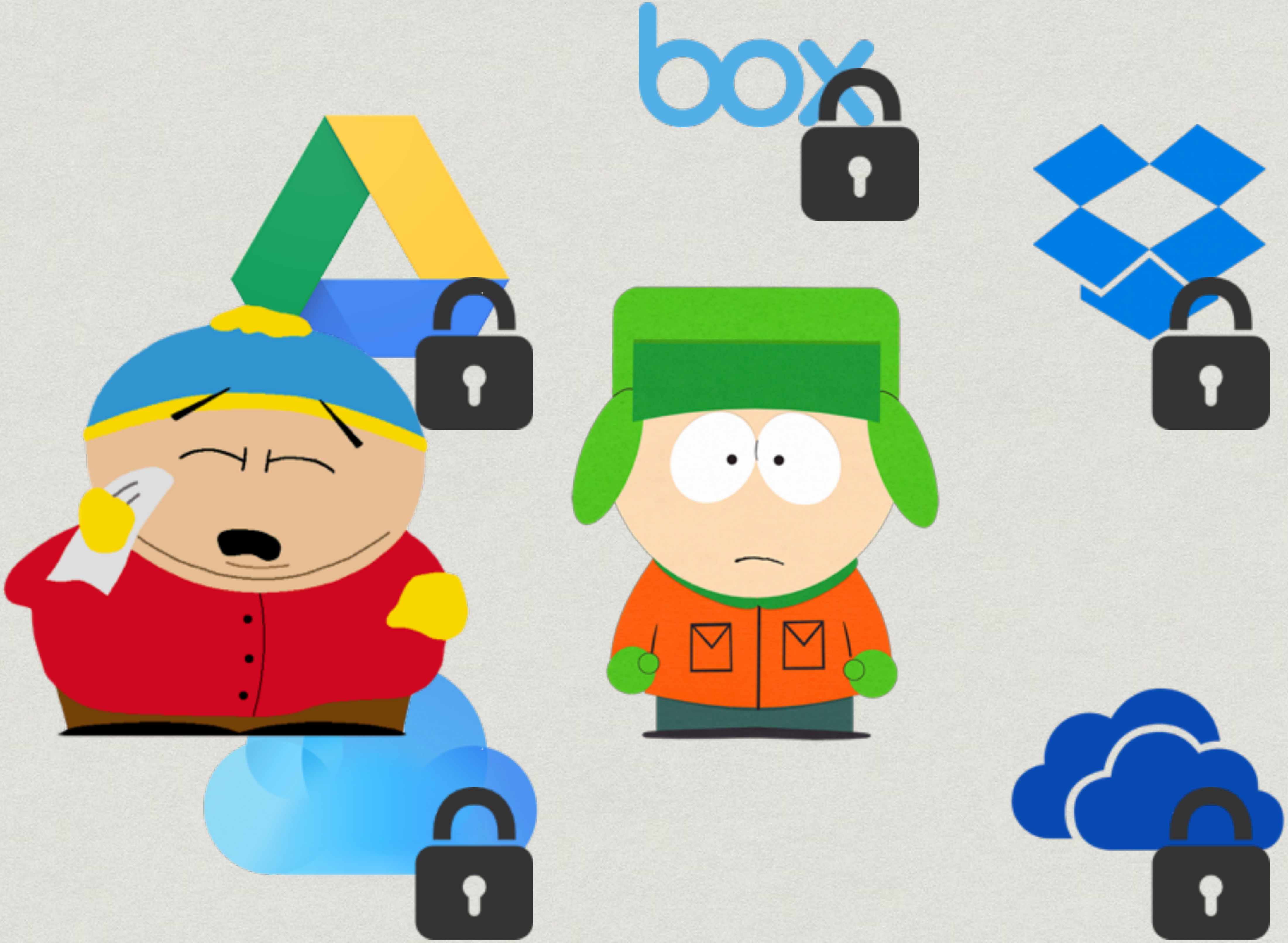
SÉMINAIRE EMSEC - RAPHAEL BOST

box

box


box


box


box

Searchable Encryption

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

Generic Solutions

We can use generic tools to solve this problem:

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

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 use less expensive solutions?

Ad-hoc Constructions

Can we use less expensive solutions?

* Yes, but …

Ad-hoc Constructions

Can we use less expensive solutions?

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

Ad-hoc Constructions

Can we use less expensive solutions?

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

Ad-hoc Constructions

Can we use less expensive solutions?

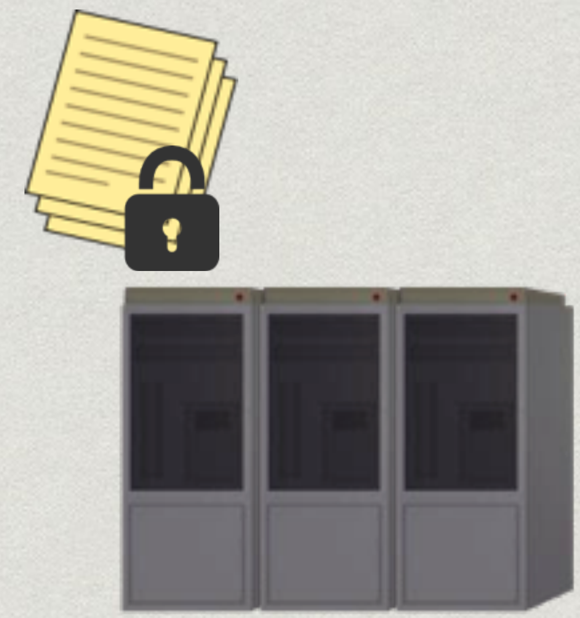
- Yes, but …
- … we have to leak some information

Security/performance tradeoff

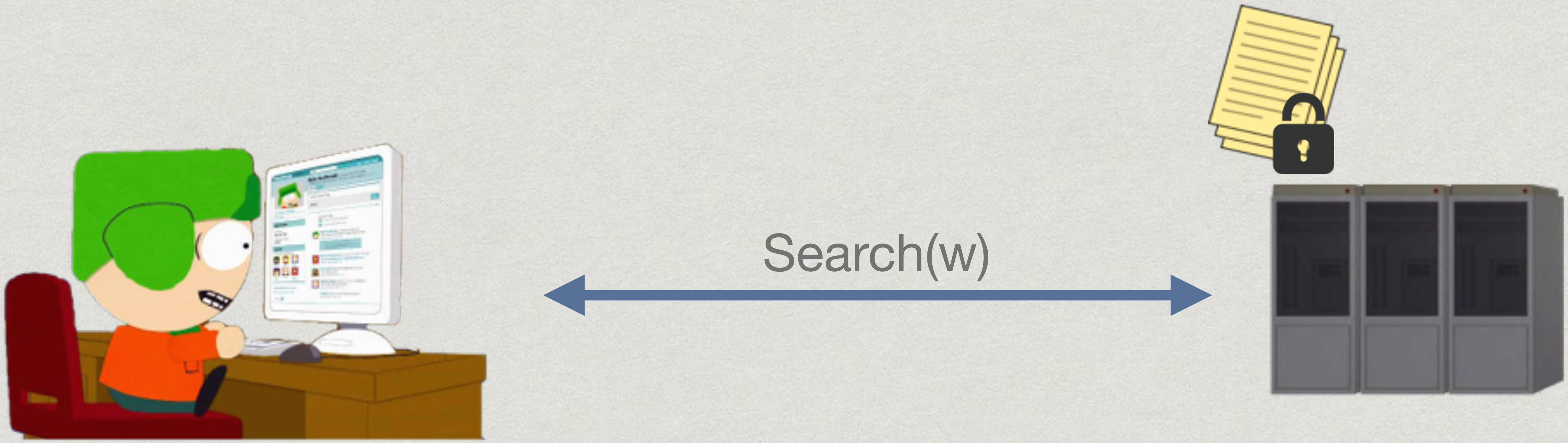
Security of SSE

Security of SSE

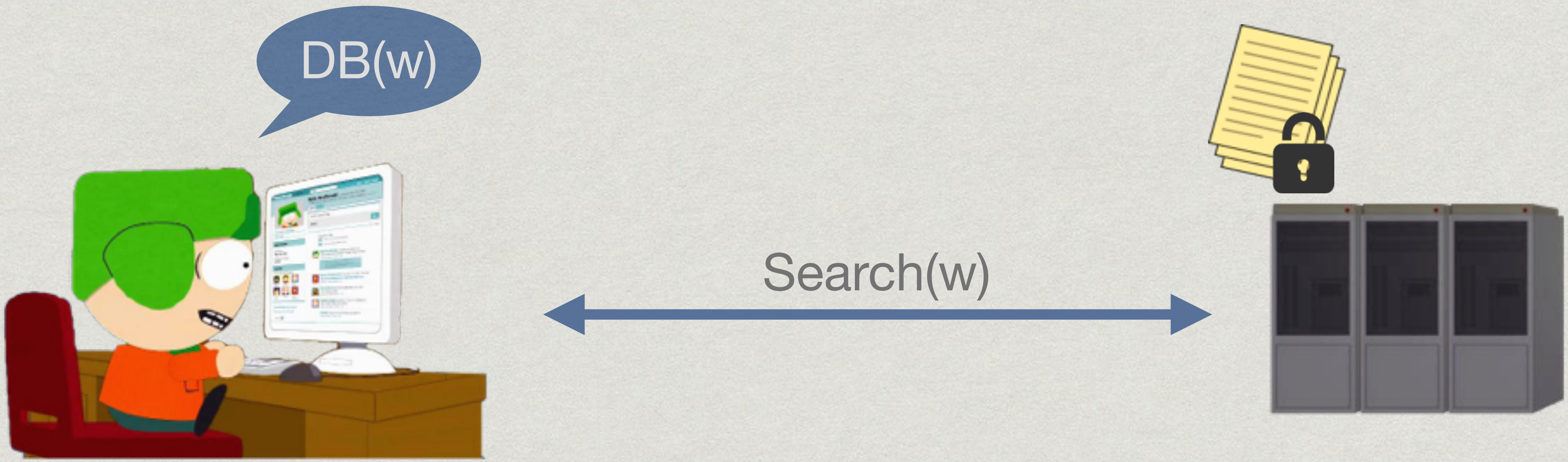
Security of SSE



Security of SSE

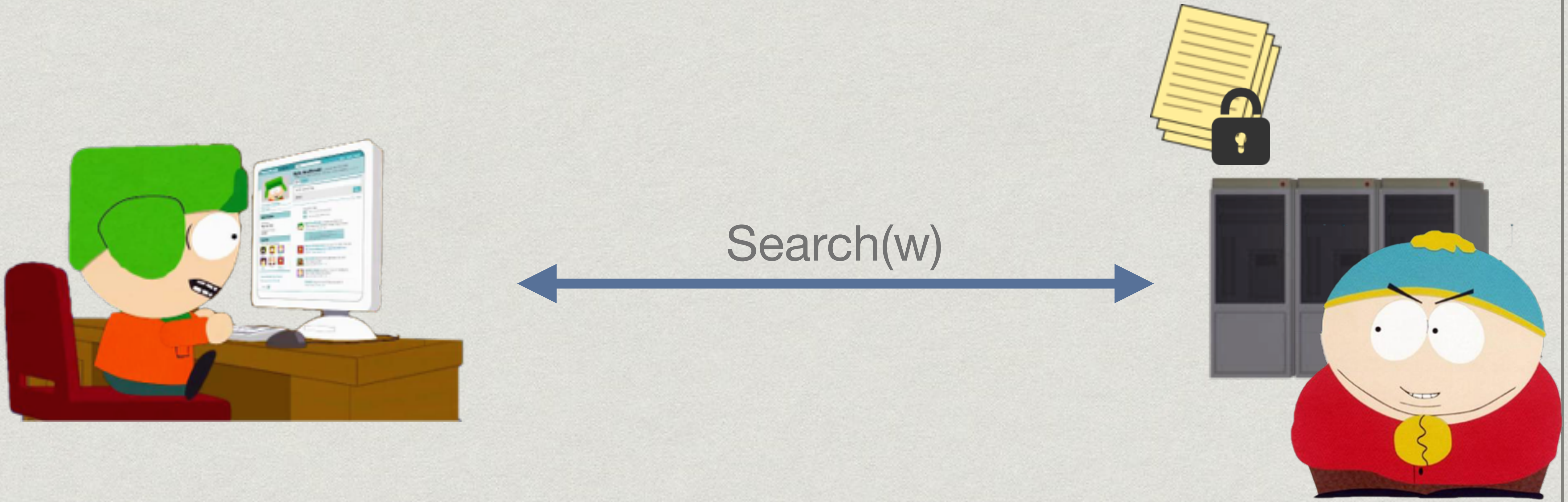


Security of SSE



Security of SSE

Security of SSE

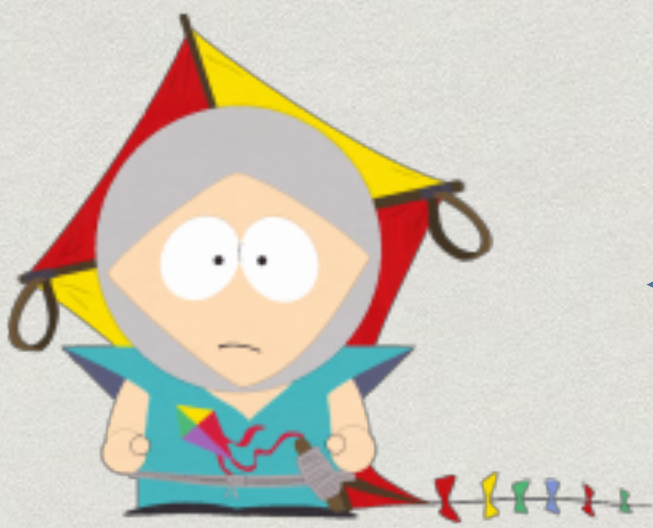


Security of SSE

Search(w)

Everything the server learns, he can compute from the leakage

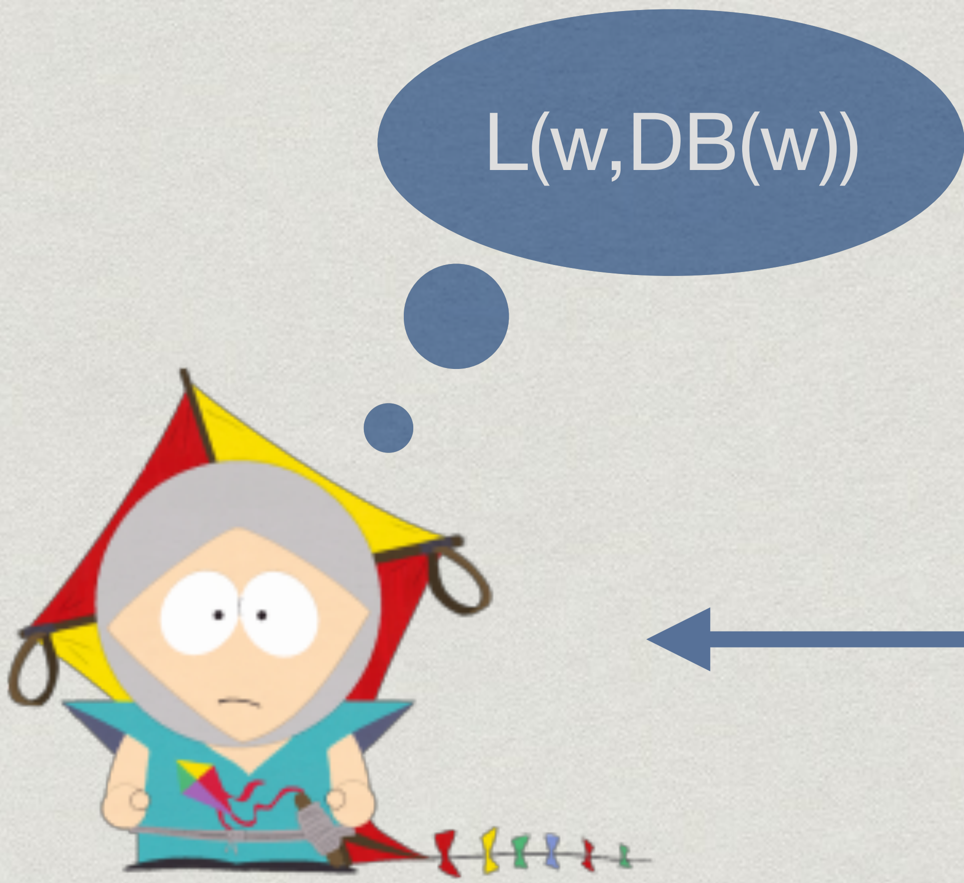
Security of SSE

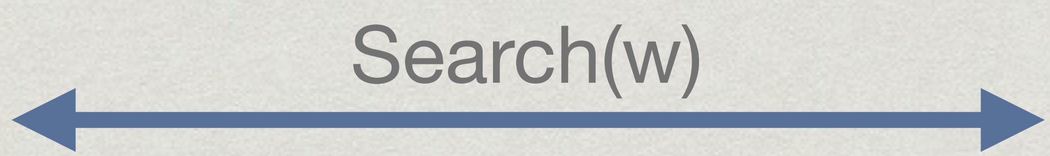


Search(w)

Everything the server learns, he can compute from the leakage

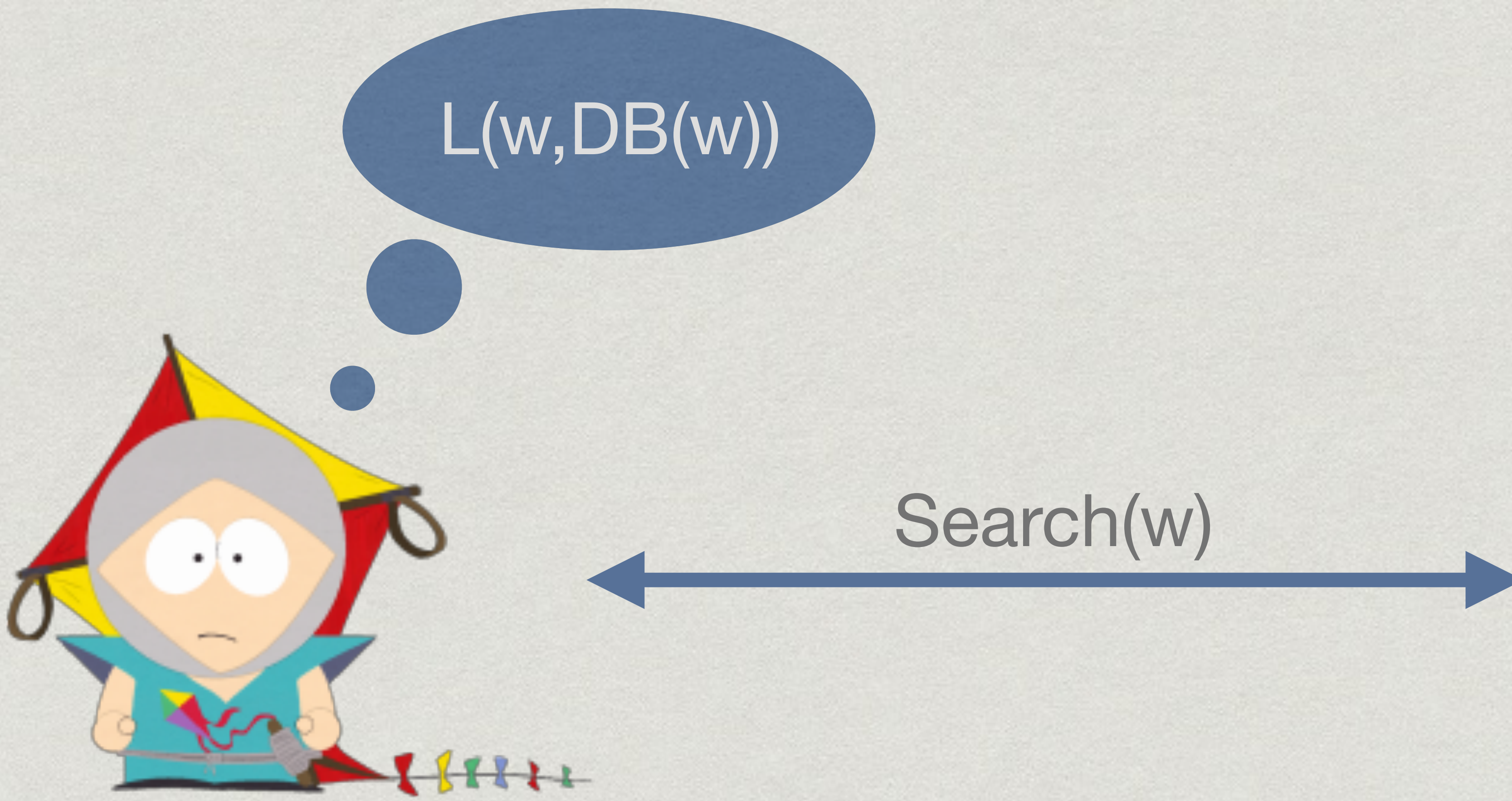
Security of SSE





Everything the server learns, he can compute from the leakage

Security of SSE



Everything the server learns, he can compute from the leakage

The search protocol can be simulated from the leakage

Leakage

Leakage

- Search leakage :
  - repetition of queries
  - results

Leakage

- Search leakage :
  - repetition of queries
  - results
- Update leakage:
  - updated documents
  - updated keywords
  - ...

Previous Results

Previous Results

* First constructions [SWP00]

Previous Results

- First constructions [SWP00]
- Formalization of the security model [CGKO06]

Previous Results

- First constructions [SWP00]
- Formalization of the security model [CGKO06]
- Efficient dynamic constructions [KPR12]

Previous Results

* First constructions [SWP00]
* Formalization of the security model [CGKO06]
* Efficient dynamic constructions [KPR12]
* Boolean queries & scalability [CJJKRS13]

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, …)

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]

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]
- …

Passive vs. Active

- (Almost) all previous results were proven secure in the honest-but-curious setting

Passive vs. Active

- (Almost) all previous results were proven secure in the honest-but-curious setting
- But we can consider a very evil adversary who modifies search results …

Passive vs. Active

- (Almost) all previous results were proven secure in the honest-but-curious setting
- But we can consider a very evil adversary who modifies search results …
- … or the database

Passive vs. Active

- (Almost) all previous results were proven secure in the honest-but-curious setting
- But we can consider a very evil adversary who modifies search results …
- … or the database

Passive vs. Active

- (Almost) all previous results were proven secure in the honest-but-curious setting
- But we can consider a very evil adversary who modifies search results …
- … or the database

Verifiable SSE

First try

- We encrypt the reversed index: we consider $\{(w, ind) \mid w \in D_{ind}\}$
- MAC each pair: $\{(w, ind \| F(w, ind)) \mid w \in D_{ind}\}$
- If the MAC is unforgeable, the server will not be able to add a false result
- Yet he still is able to remove one result …
- MAC DB(w) and return it for each search query

Dynamic scheme

Let us consider the following operations:

Dynamic scheme

Let us consider the following operations:

- Kyle searches w, the server retrieves DB(w) and MAC(DB(w))

Dynamic scheme

Let us consider the following operations:

- Kyle searches w, the server retrieves DB(w) and MAC(DB(w))
- Kyle updates the DB, on keyword w

Dynamic scheme

Let us consider the following operations:

- Kyle searches w, the server retrieves DB(w) and MAC(DB(w))
- Kyle updates the DB, on keyword w
- Kyle searches w again. Instead of DB’(w), the server returns the old DB(w) and MAC(DB(w))

Dynamic scheme

Let us consider the following operations:

- Kyle searches w, the server retrieves DB(w) and MAC(DB(w))
- Kyle updates the DB, on keyword w
- Kyle searches w again. Instead of DB'(w), the server returns the old DB(w) and MAC(DB(w))

MACs are ineffective against replay attacks

Memory Checking

- A user with limited storage abilities wants to maintain a large DB on a remote untrusted server
- How many queries does the client have to make to the untrusted server per request
- Lower bound: $\Omega(\log n / \log \log n)$ (for n blocks)

Lower Bound on VSSE

- We can write a reduction from memory checkers to VSSE
  - Encode each block as a document index, each block address as a keyword
  - Block access: Search
  - Block update: Delete old index, add new index

Lower bound: $\Omega(\log |W| / \log \log |W|)$

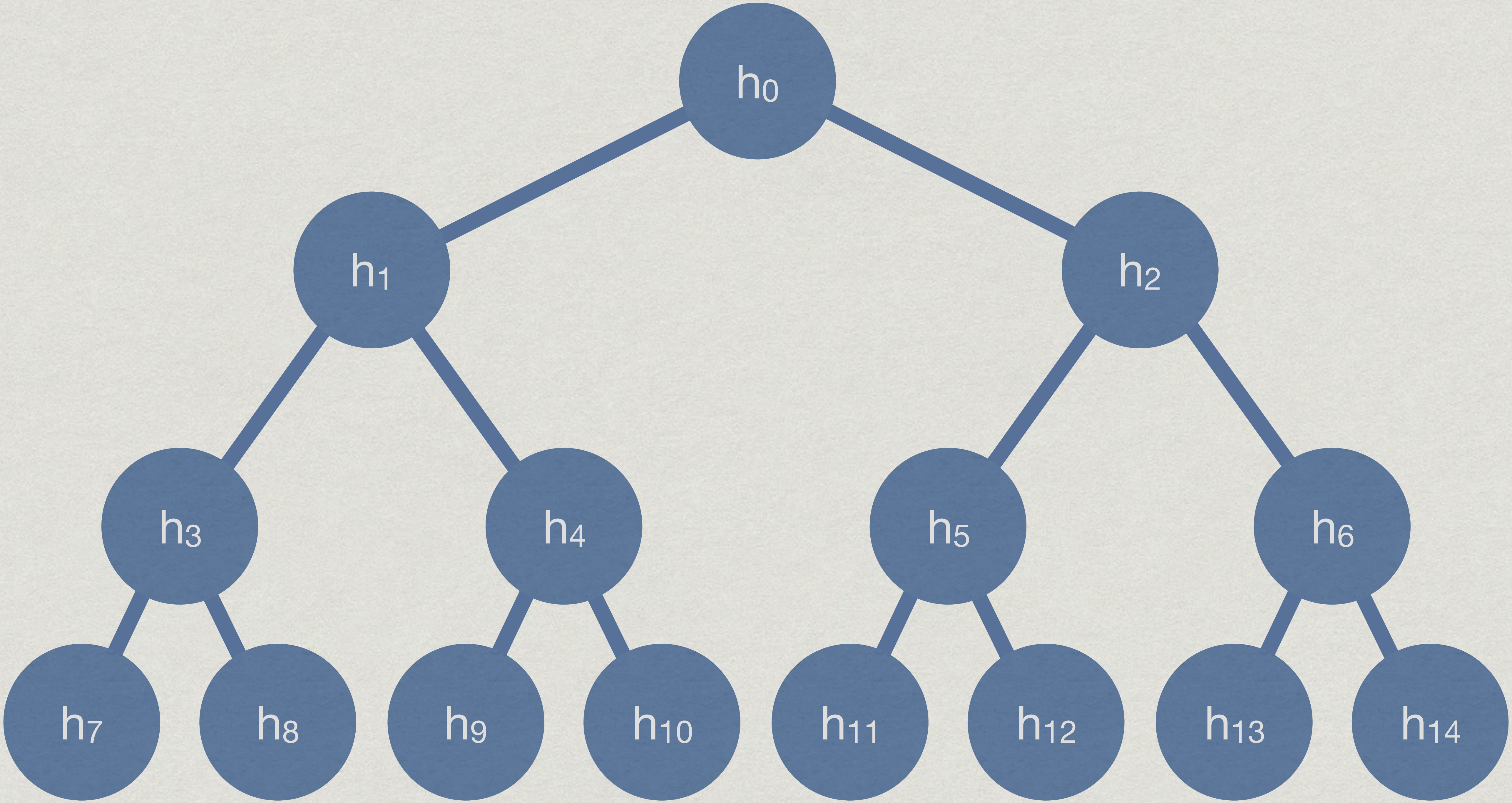
Verifiable Map

Outsource a map data structure.
Two types of constructions

- Hash based: Merkle Hash tree
  Query and updates in log(n) (optimal)

- Accumulator based
  Query in O(1), Updates in $O(n^{\varepsilon})$ or
  Query in $O(n^{\varepsilon})$, Updates in O(1)

Merkle Hash Tree



$$h_i = H(h_{2i+1} \,||\, h_{2i+2})$$

Cryptographic Accumulator

- Short membership proof
- E.g. $E = \{r_1, \ldots, r_n\}$ set of k bits primes
  $f(E) = g^{r_1 \cdots r_n} \bmod N$ where $g \in QR_N$ and N $k' > k$ bits RSA modulus
- Can be built from other assumptions (DHE, BM)

Where are we now?

- For each $w \in W$, Kyle stores MAC(DB(w))
- This map is outsourced using a verifiable map
- How to update?
  - Recompute MAC(DB(w)) every time it is modified
  - Incremental MAC/Hash

(Multi)Set Hashing

- Input: (multi)set, output: a string whose value is independent of the elements' order
- Incremental: $H(M \cup M') = H(M) \diamond H(M')$ for some $\diamond$
- Collision resistance: hard to construct $M_1 \neq M_2$ s.t. $H(M_1) = H(M_2)$

Set Hashing Constructions

- General idea: let G be a group with generator g
- $H(M) = \prod_{a \in M} g^{m(a)h(a)}$
- Examples for G: $(Z_N, \times)$, $(Z_N, +)$, EC

Our generic construction

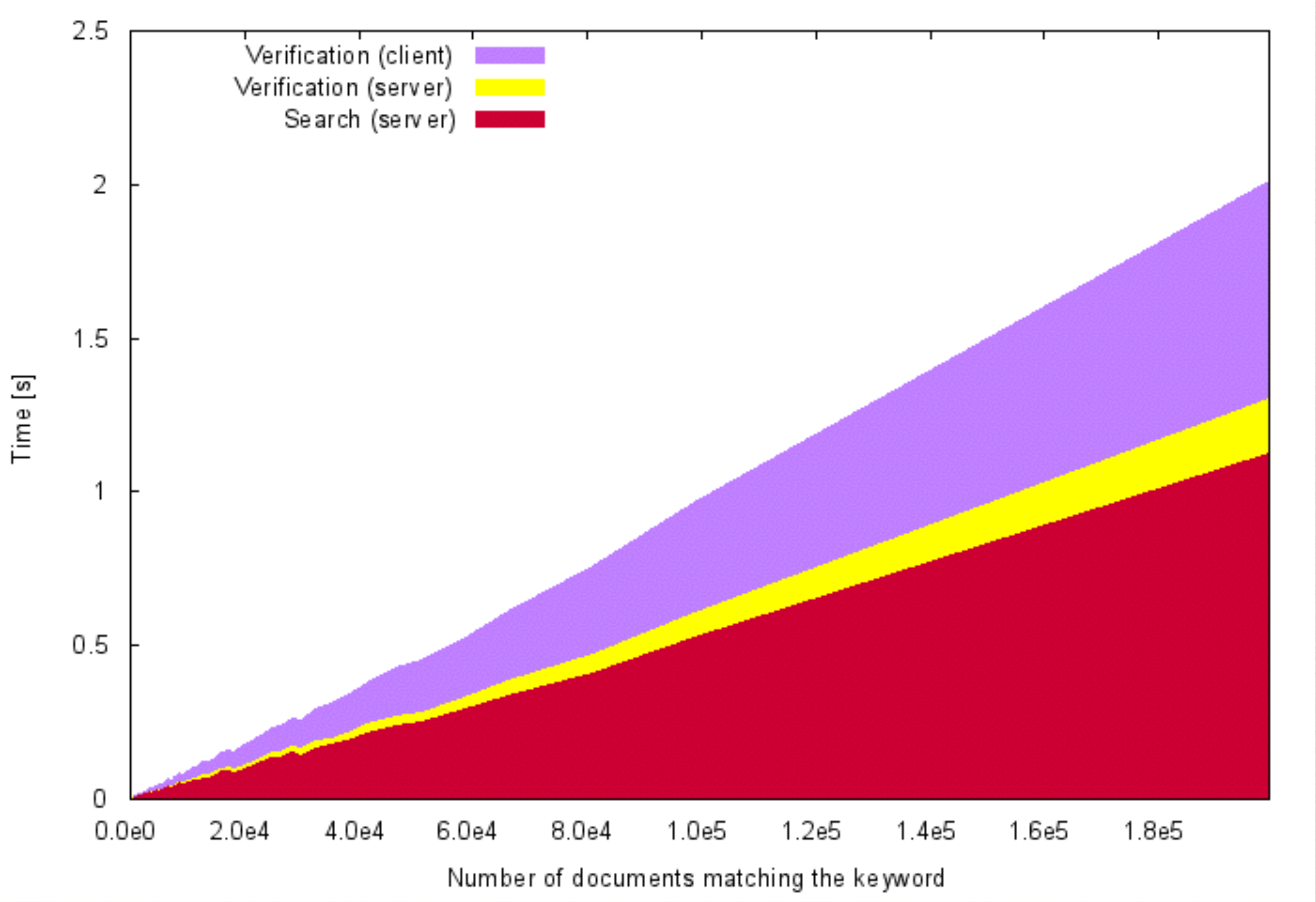
- Set hash DB(w) for all w
- Put the results in a verifiable map
- When searching, get the hash from the VM, check it matches the awaited value
- When updating, incrementally update the hash of DB(w) in the map

Complexity

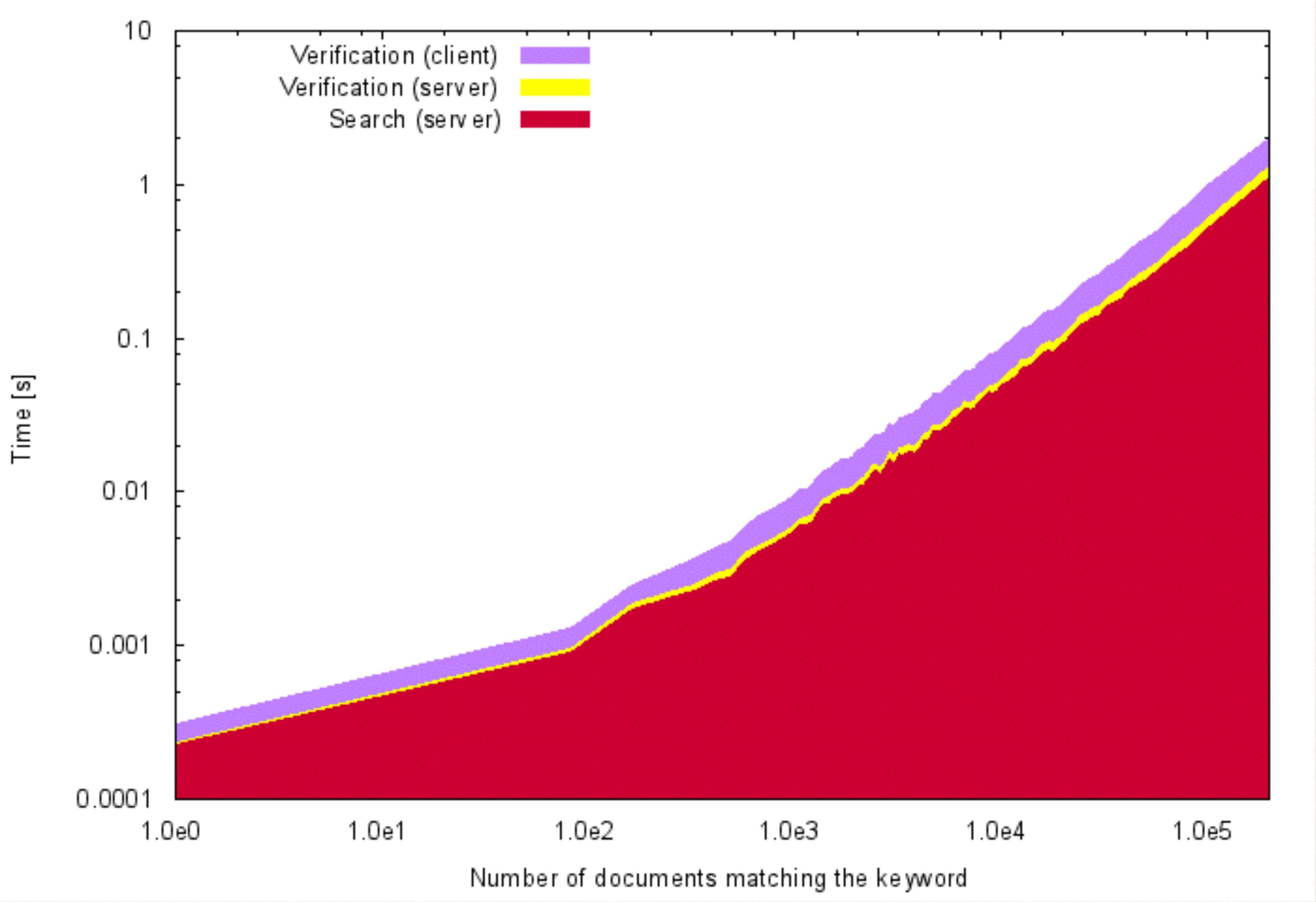
- Hash-based map:
  Search: $O(\log |W| + m)$, Update: $O(\log |W|)$
- Accumulator-based map v1:
  Search: $O(m)$, Update: $O(|W|^{\varepsilon})$
- Accumulator-based map v2:
  Search: $O(m + |W|^{\varepsilon})$, Update: $O(1)$

Optimal with 3 different meanings

Implementation



Implementation



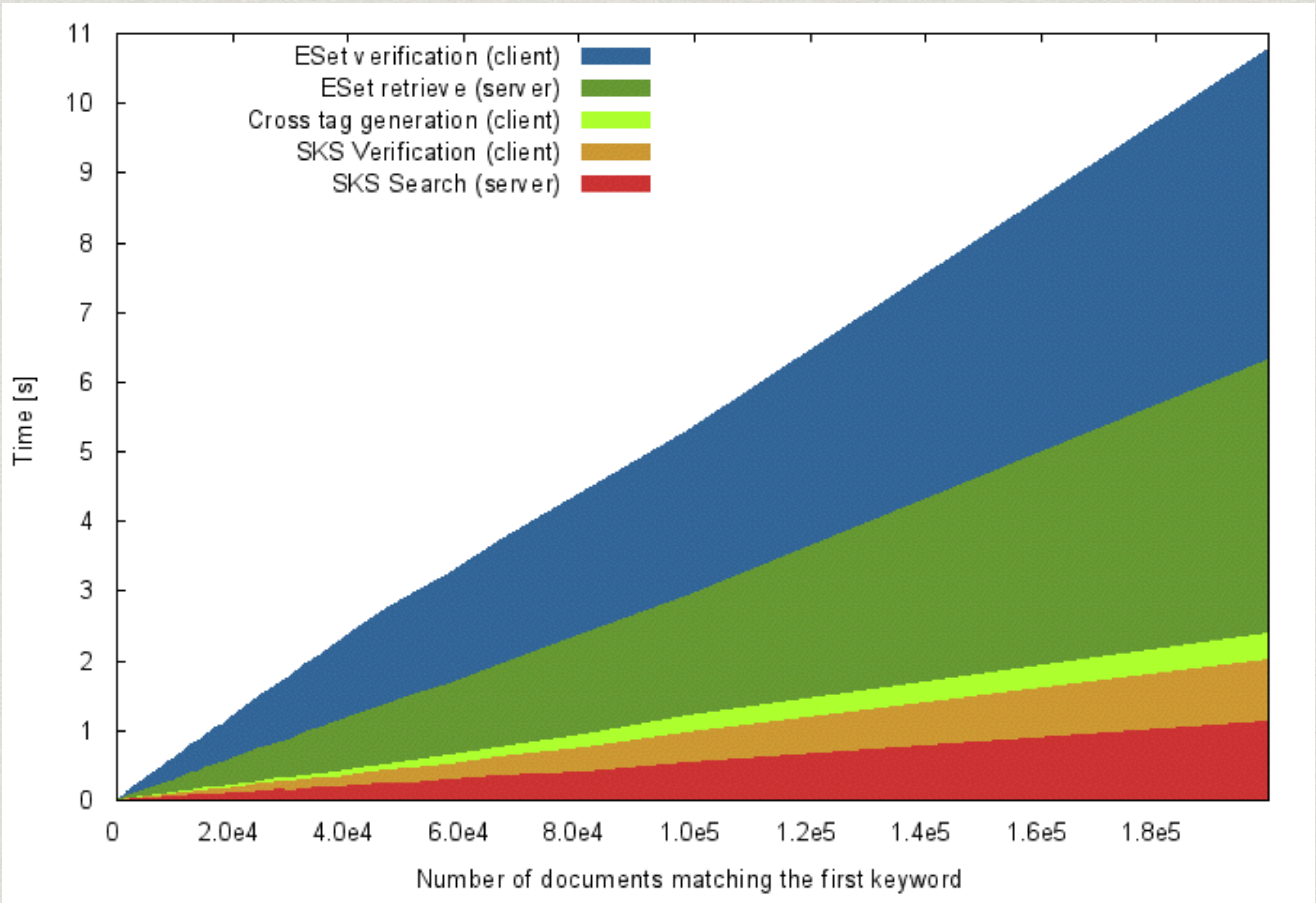
Multiple Keywords

- Cash et al., CRYPTO'13: search $DB(s \wedge x)$
  Generate DB(s), and for all ind $\in$ DB(s), look if $x \in D_{ind}$
- To verify, first verify DB(s), then generate proofs for the proposition $x \in D_{ind}$ (or $x \notin D_{ind}$)

Multiple Keywords

- Cash et al., CRYPTO'13: search DB($s \wedge x$)
  Generate DB(s), and for all ind $\in$ DB(s), look if $x \in D_{ind}$
- To verify, first verify DB(s), then generate proofs for the proposition $x \in D_{ind}$ (or $x \notin D_{ind}$)

Multiple Keywords



Next challenges

- Improve multiple keywords verification (batch verifications ??)
- Forward privacy
- Better understanding of leakage, avoid leakage abuse attacks