The Locality of Searchable Symmetric Encryption

David Cash

Rutgers

Stefano Tessaro

UCSB

Connecting **security** and **i/o efficiency**

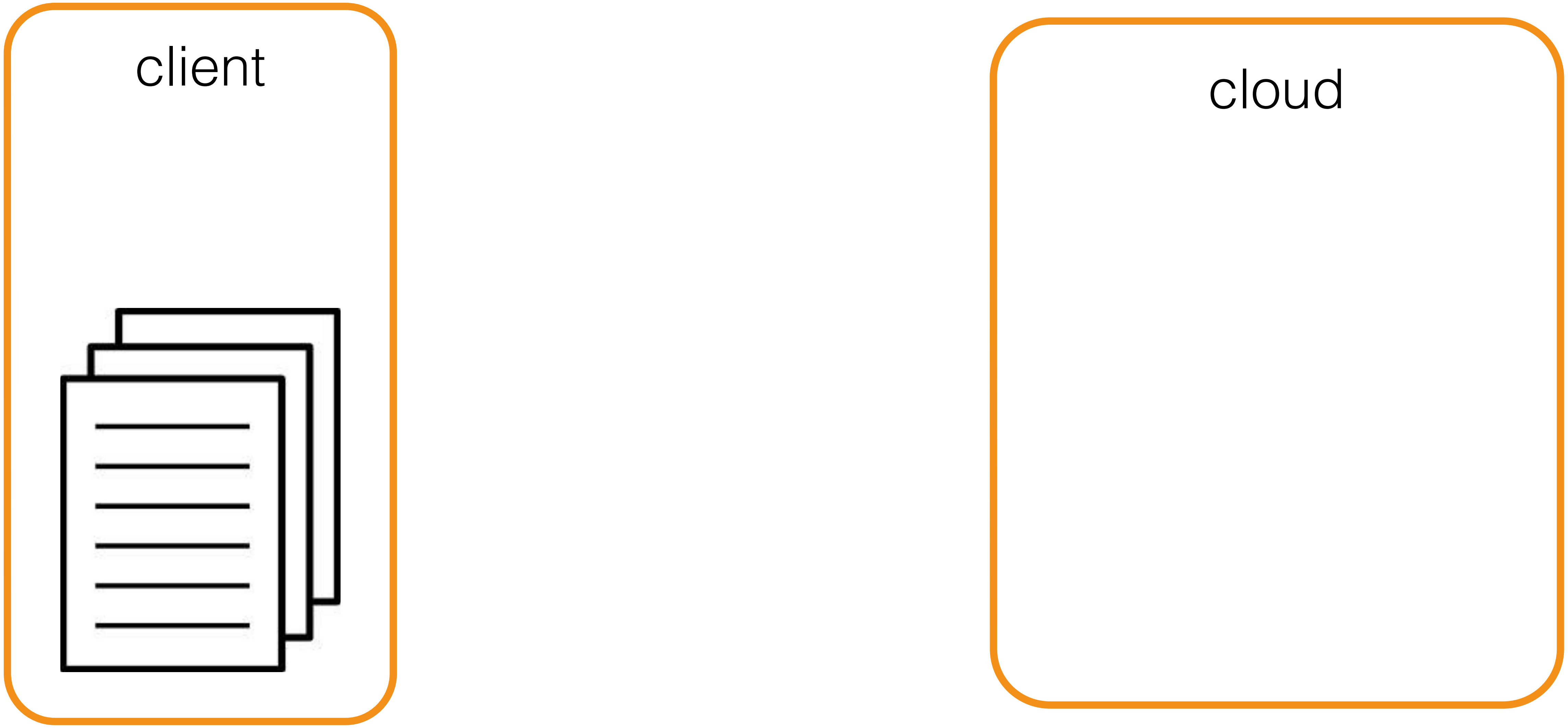
or

How a security notion can force inefficient disk utilization when encrypting large files

➡ new type of lower bound

➡ new constructions

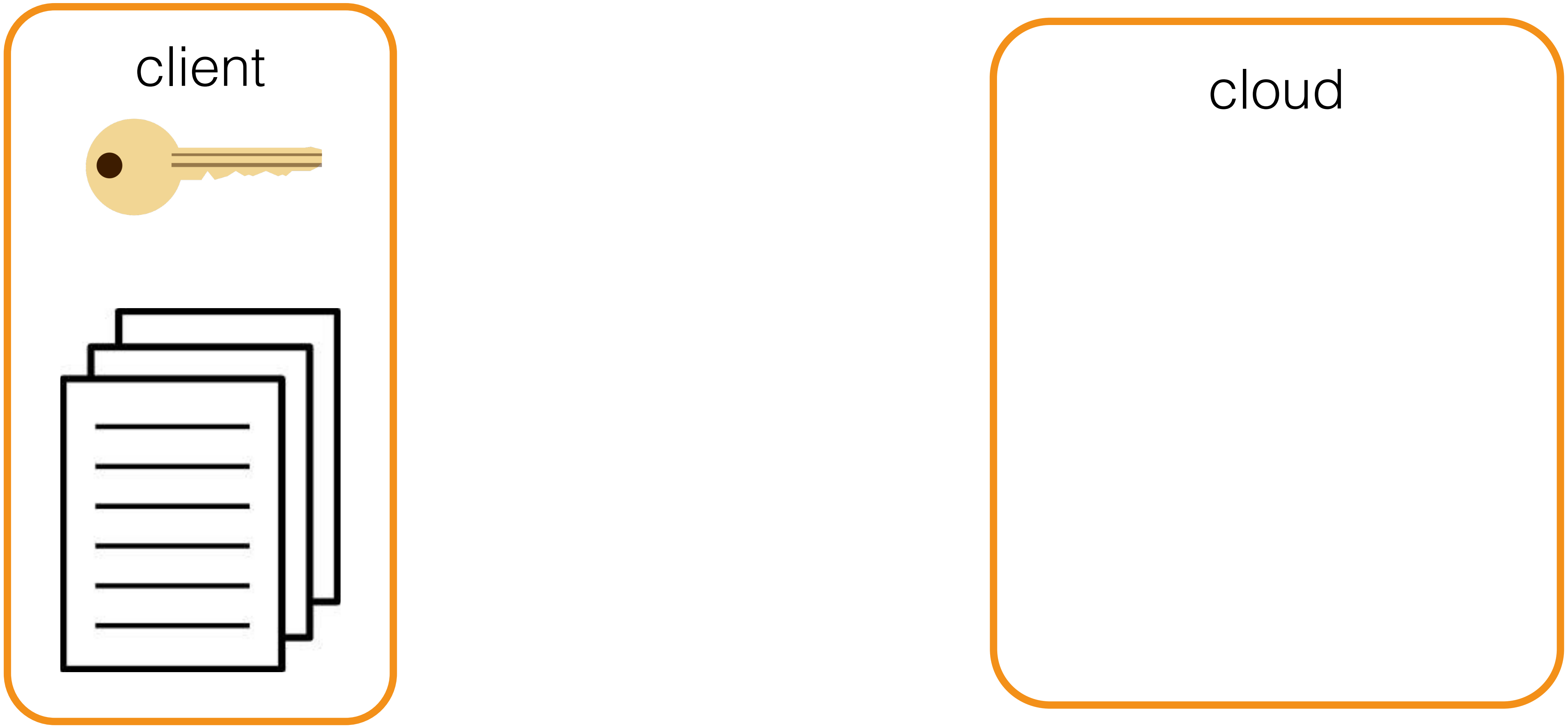
remotely storing encrypted documents



client

cloud

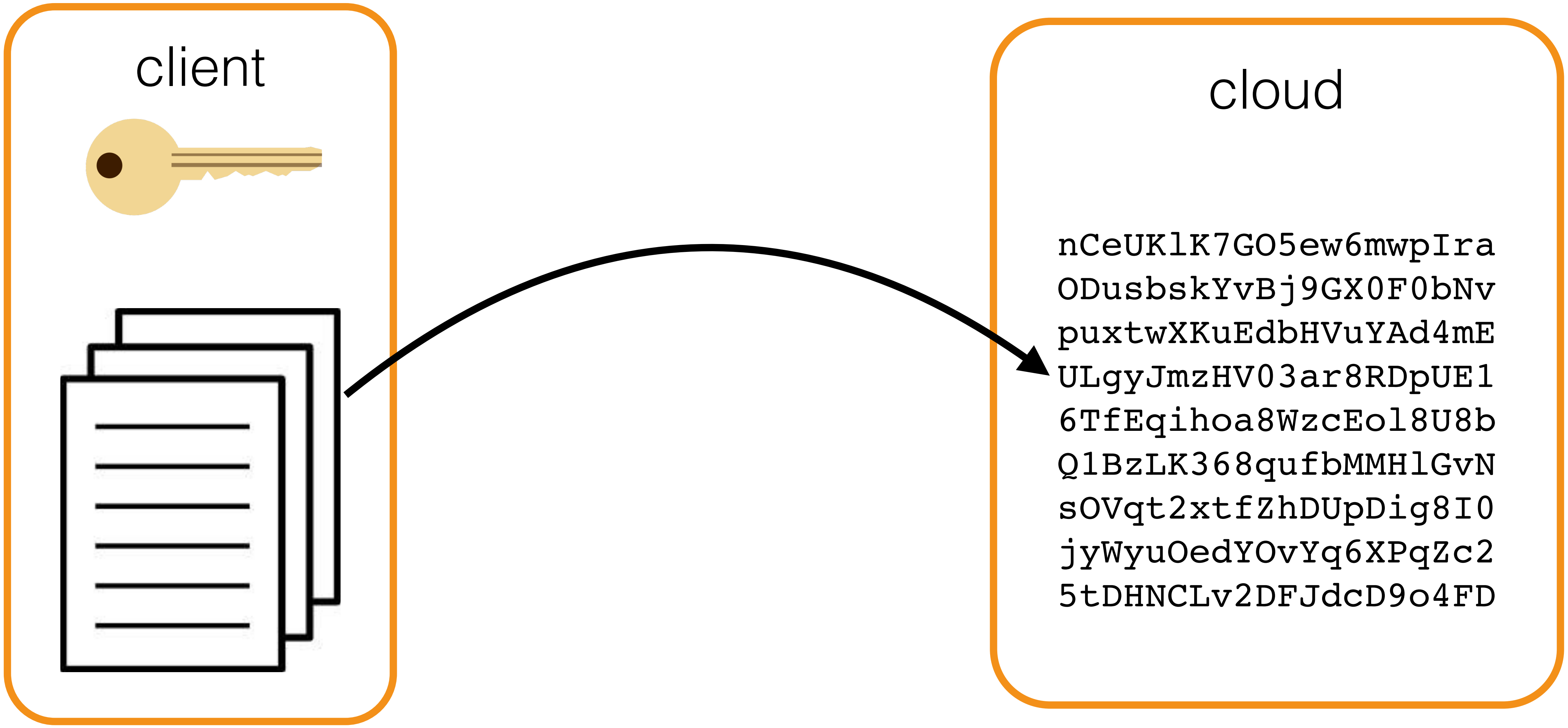
remotely storing encrypted documents



client

cloud

remotely storing encrypted documents

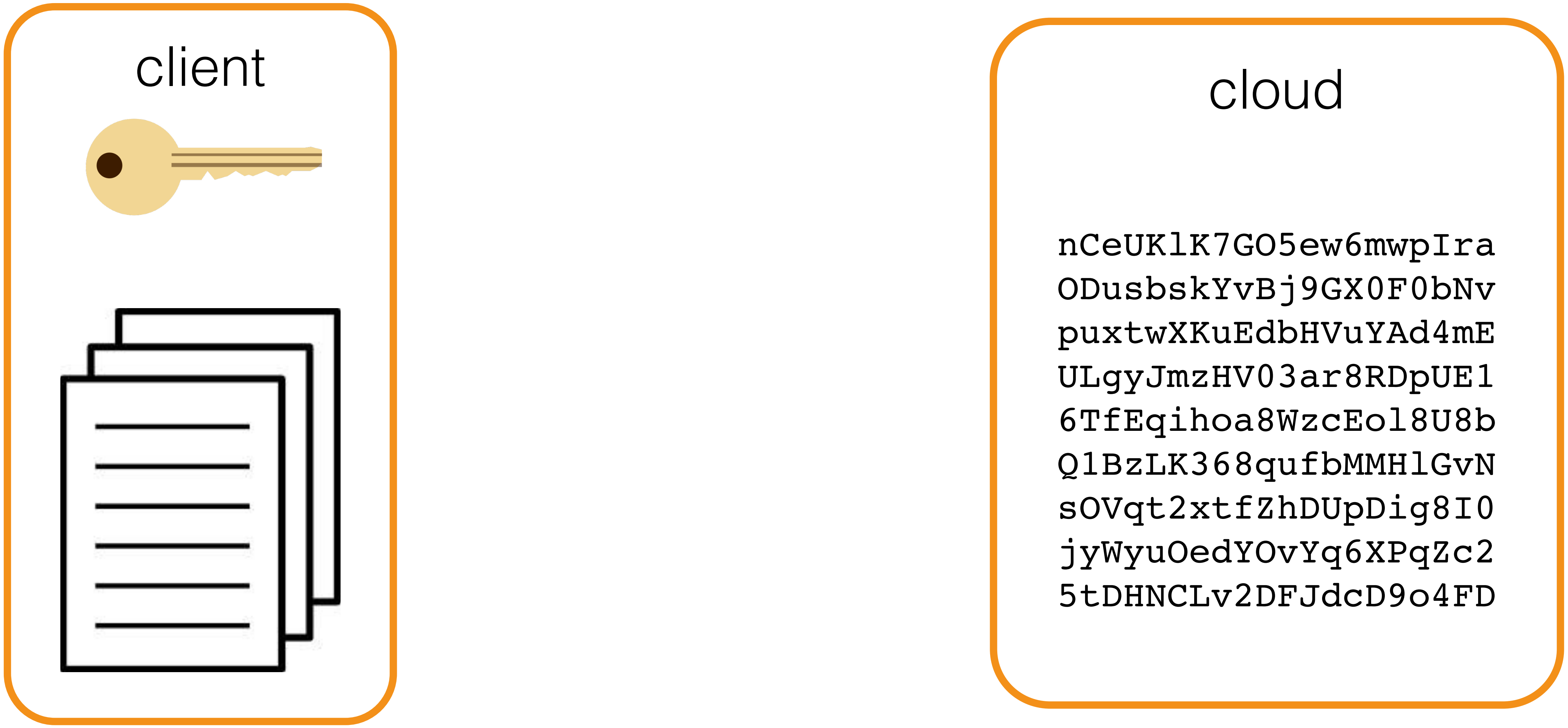


client

cloud

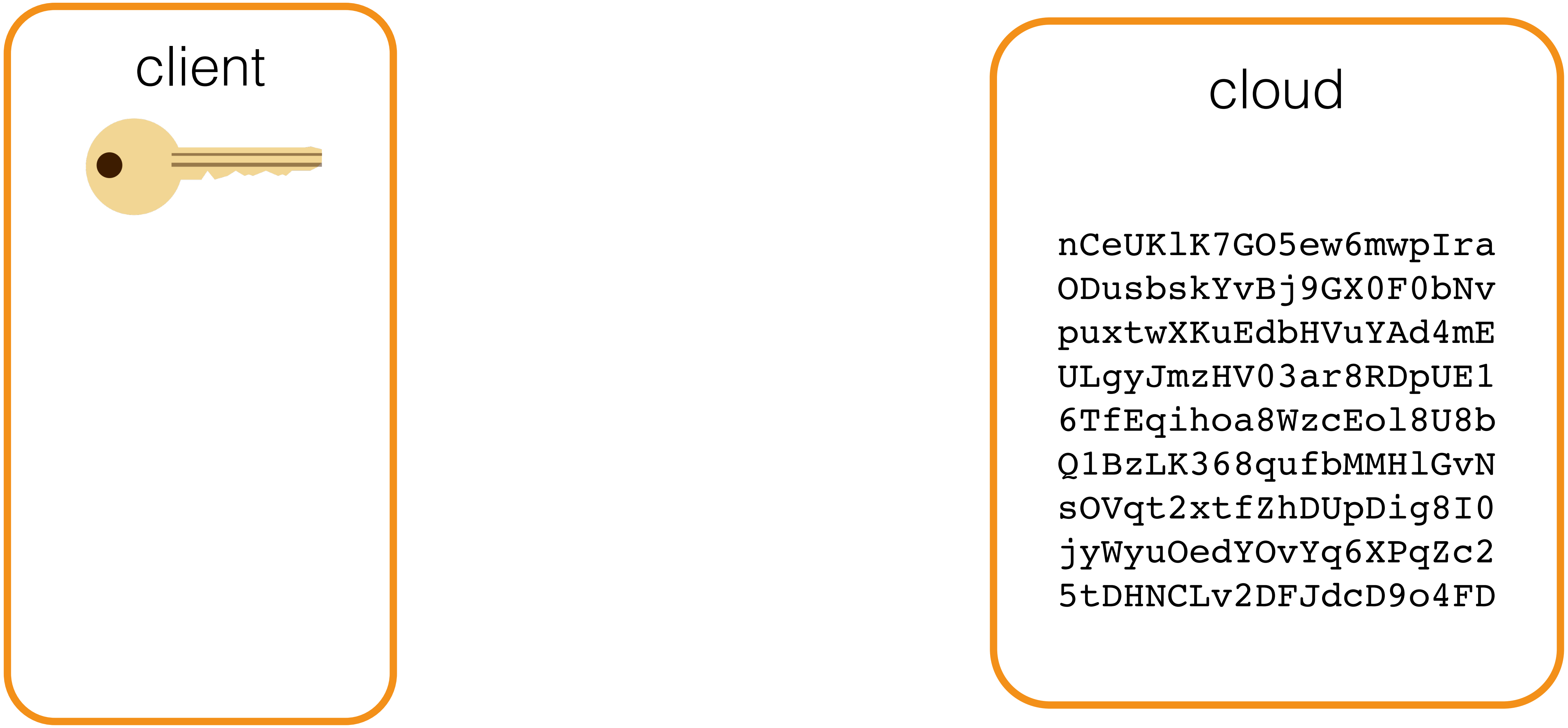
```
nCeUKlK7GO5ew6mwpIra
ODusbskYvBj9GX0F0bNv
puxtwXKuEdbHVuYAd4mE
ULgyJmzHV03ar8RDpUE1
6TfEqihoa8WzcEol8U8b
QlBzLK368qufbMMHlGvN
sOVqt2xtfZhDUpDig8I0
jyWyuOedYOvYq6XPqZc2
5tDHNCLv2DFJdcD9o4FD
```

remotely storing encrypted documents



```
nCeUKlK7GO5ew6mwpIra
ODusbskYvBj9GX0F0bNv
puxtwXKuEdbHVuYAd4mE
ULgyJmzHV03ar8RDpUE1
6TfEqihoa8WzcEol8U8b
Q1BzLK368qufbMMHlGvN
sOVqt2xtfZhDUpDig8I0
jyWyuOedYOvYq6XPqZc2
5tDHNCLv2DFJdcD9o4FD
```

remotely storing encrypted documents

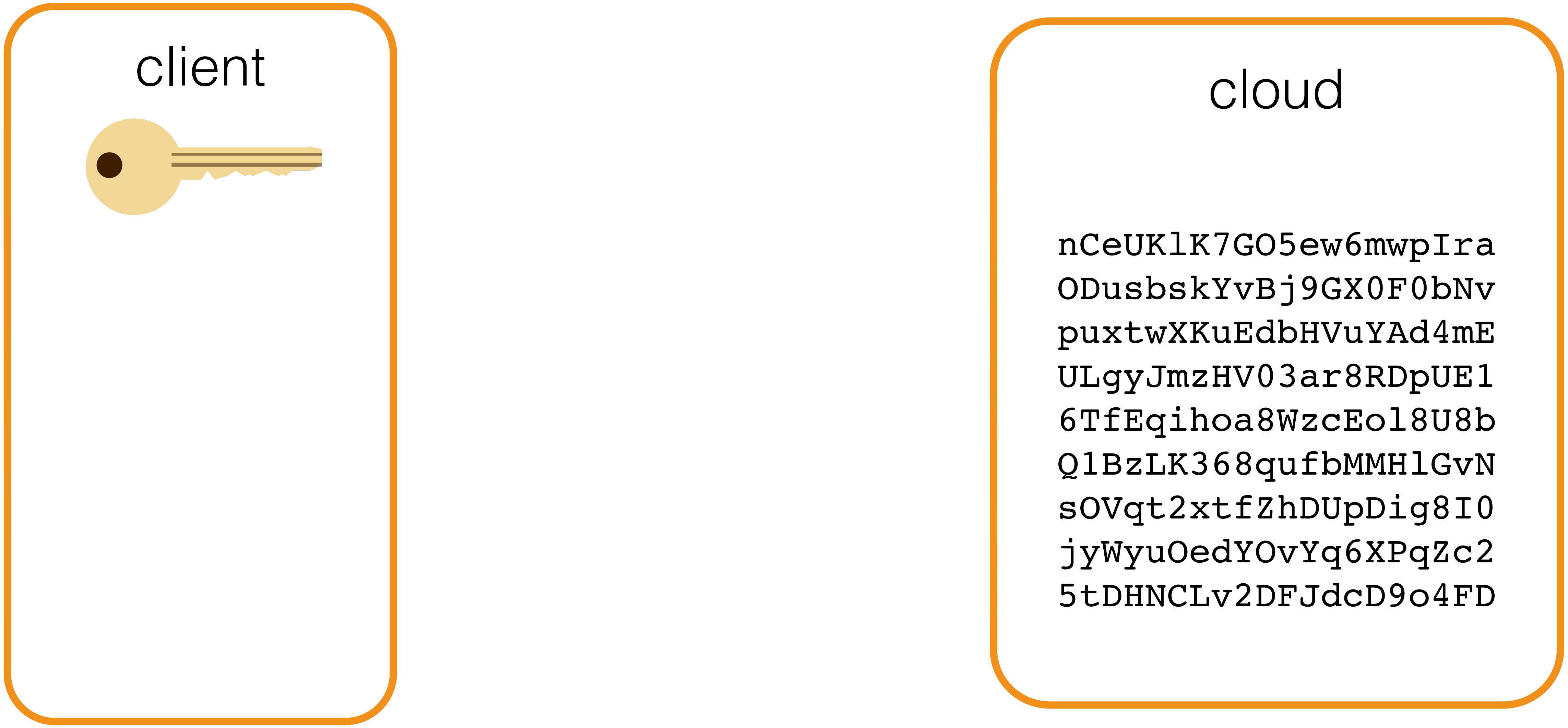


client

cloud

```
nCeUKlK7GO5ew6mwpIra
ODusbskYvBj9GX0F0bNv
puxtwXKuEdbHVuYAd4mE
ULgyJmzHV03ar8RDpUE1
6TfEqihoa8WzcEol8U8b
Q1BzLK368qufbMMHlGvN
sOVqt2xtfZhDUpDig8I0
jyWyuOedYOvYq6XPqZc2
5tDHNCLv2DFJdcD9o4FD
```

remotely storing encrypted documents



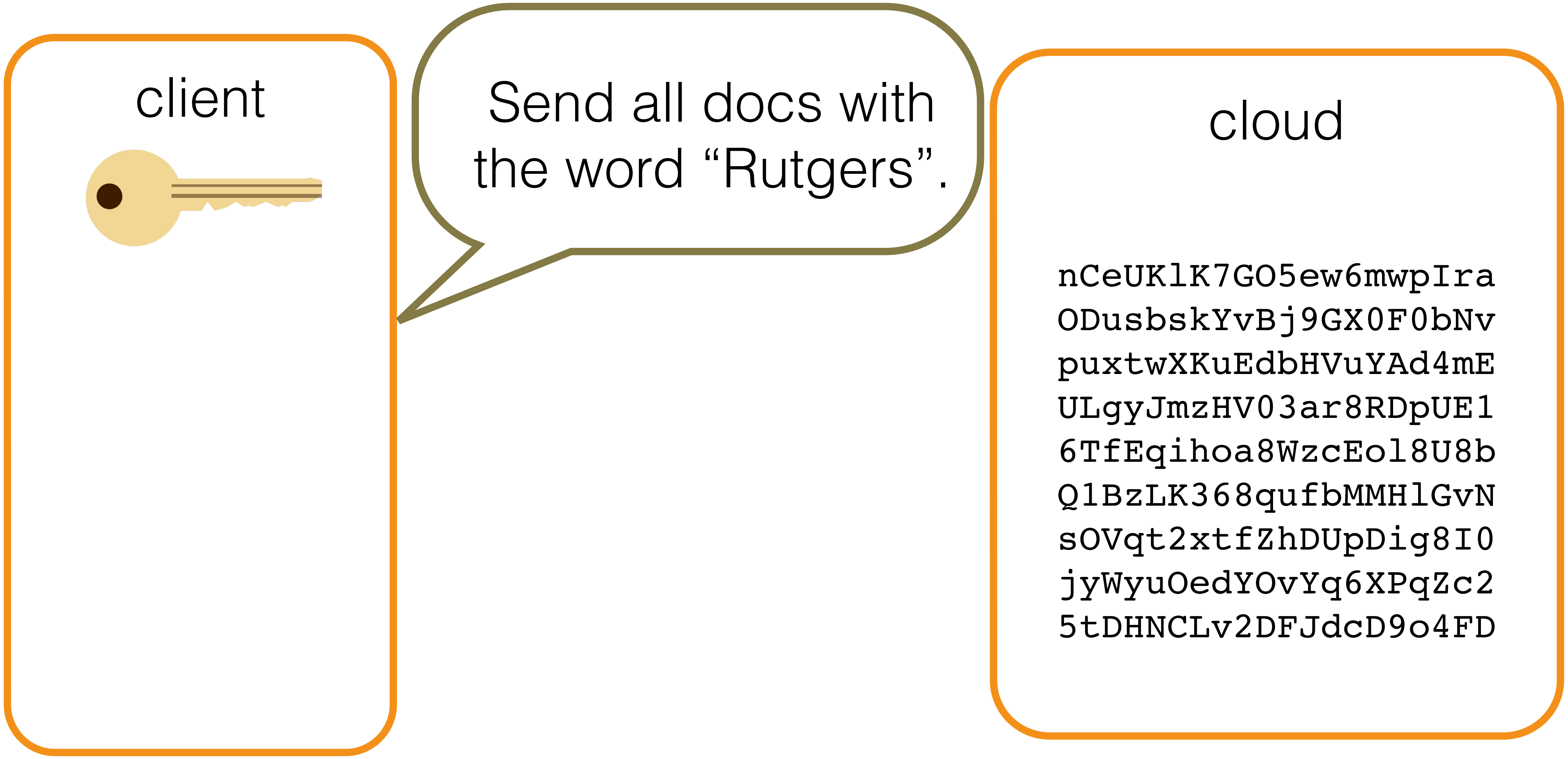
client

cloud

```
nCeUKlK7GO5ew6mwpIra
ODusbskYvBj9GX0F0bNv
puxtwXKuEdbHVuYAd4mE
ULgyJmzHV03ar8RDpUE1
6TfEqihoa8WzcEol8U8b
Q1BzLK368qufbMMHlGvN
sOVqt2xtfZhDUpDig8I0
jyWyuOedYOvYq6XPqZc2
5tDHNCLv2DFJdcD9o4FD
```

➡ client encryption prevents server from helping by indexing, searching, organizing, …

remotely storing encrypted documents



client

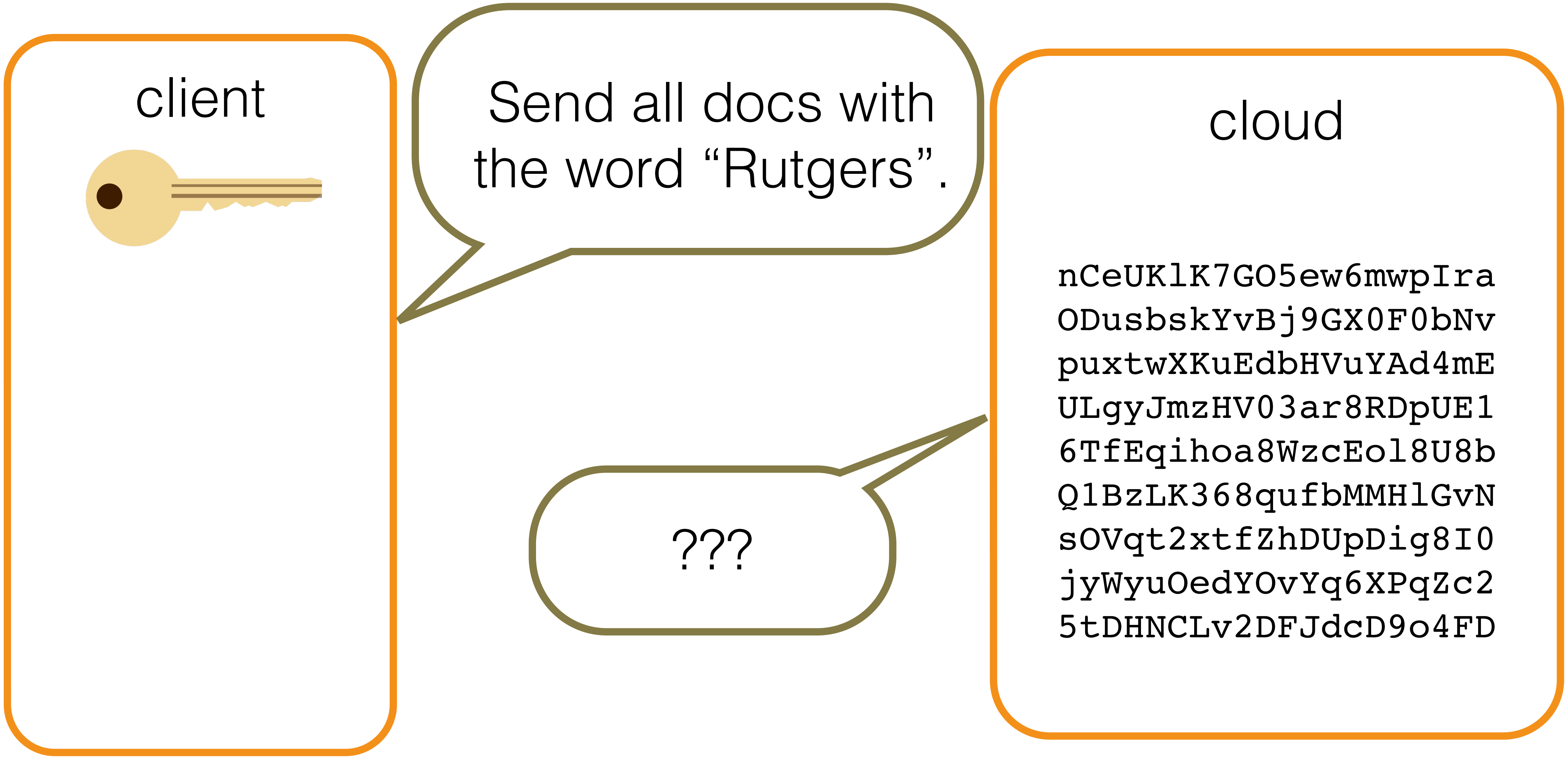
Send all docs with the word “Rutgers”.

cloud

```
nCeUKlK7GO5ew6mwpIra
ODusbskYvBj9GX0F0bNv
puxtwXKuEdbHVuYAd4mE
ULgyJmzHV03ar8RDpUE1
6TfEqihoa8WzcEol8U8b
Q1BzLK368qufbMMHlGvN
sOVqt2xtfZhDUpDig8I0
jyWyuOedYOvYq6XPqZc2
5tDHNCLv2DFJdcD9o4FD
```

➡ client encryption prevents server from helping by indexing, searching, organizing, …

remotely storing encrypted documents



client

Send all docs with the word “Rutgers”.

???

cloud

```
nCeUKlK7GO5ew6mwpIra
ODusbskYvBj9GX0F0bNv
puxtwXKuEdbHVuYAd4mE
ULgyJmzHV03ar8RDpUE1
6TfEqihoa8WzcEol8U8b
Q1BzLK368qufbMMHlGvN
sOVqt2xtfZhDUpDig8I0
jyWyuOedYOvYq6XPqZc2
5tDHNCLv2DFJdcD9o4FD
```

➡ client encryption prevents server from helping by indexing, searching, organizing, …

theory solution: computation on encrypted data

- homomorphic encryption
- private information retrieval
- secure multiparty computation
- oblivious RAM
- …

theory solution: computation on encrypted data

- homomorphic encryption
- private information retrieval
- secure multiparty computation
- oblivious RAM
- …

➡ enable searching w/o decryption

theory solution: computation on encrypted data

- homomorphic encryption
- private information retrieval
- secure multiparty computation
- oblivious RAM
- …

➡ enable searching w/o decryption

➡ minimal "leakage" to server

- hide doc plaintexts, query values, even which docs are downloaded

theory solution: computation on encrypted data

- homomorphic encryption
- private information retrieval
- secure multiparty computation
- oblivious RAM
- …

➡ enable searching w/o decryption

➡ minimal "leakage" to server

- hide doc plaintexts, query values, even which docs are downloaded

➡ none currently in use for encrypted search

[Song, Wagner, Perrig] & [Curtmola, Garay, Kamara, Ostrovsky]:

Different approach to encrypted search:

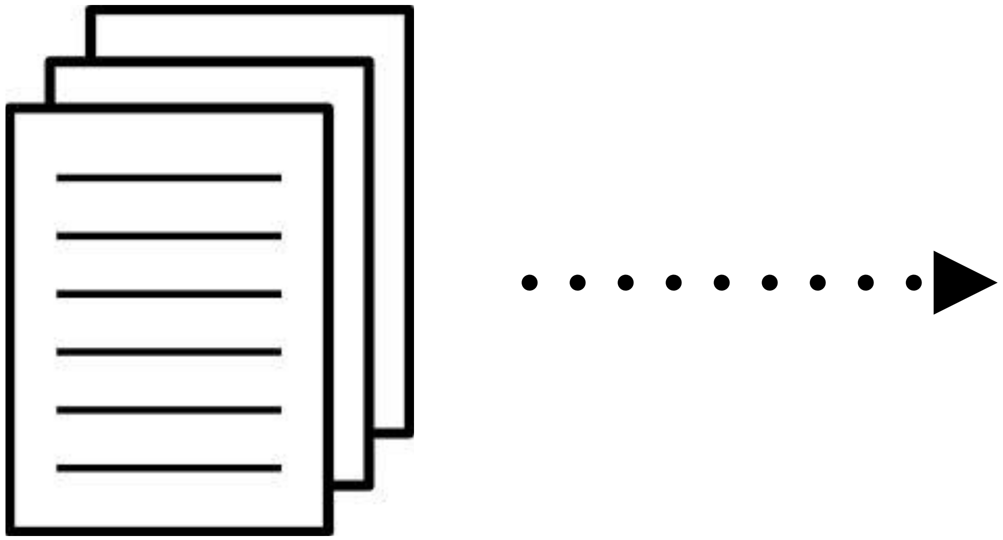
- Almost as efficient as unencrypted search
- Target weaker security - “leak the results”

- implementable - use only AES/HMAC/etc + data structures

[SWP] with [CGKO] refinement:

➡ Encrypt actual files using regular encryption

➡ Build and encrypt “(inverted) index” then delegate decryption of it later



Keyword	Doc IDs
“Rutgers”	4,9,37
“Admissions”	9,37,93,94,95,96
“Committee”	8,37,93,94
“Accept”	2,37,62,75

information retrieval terminology

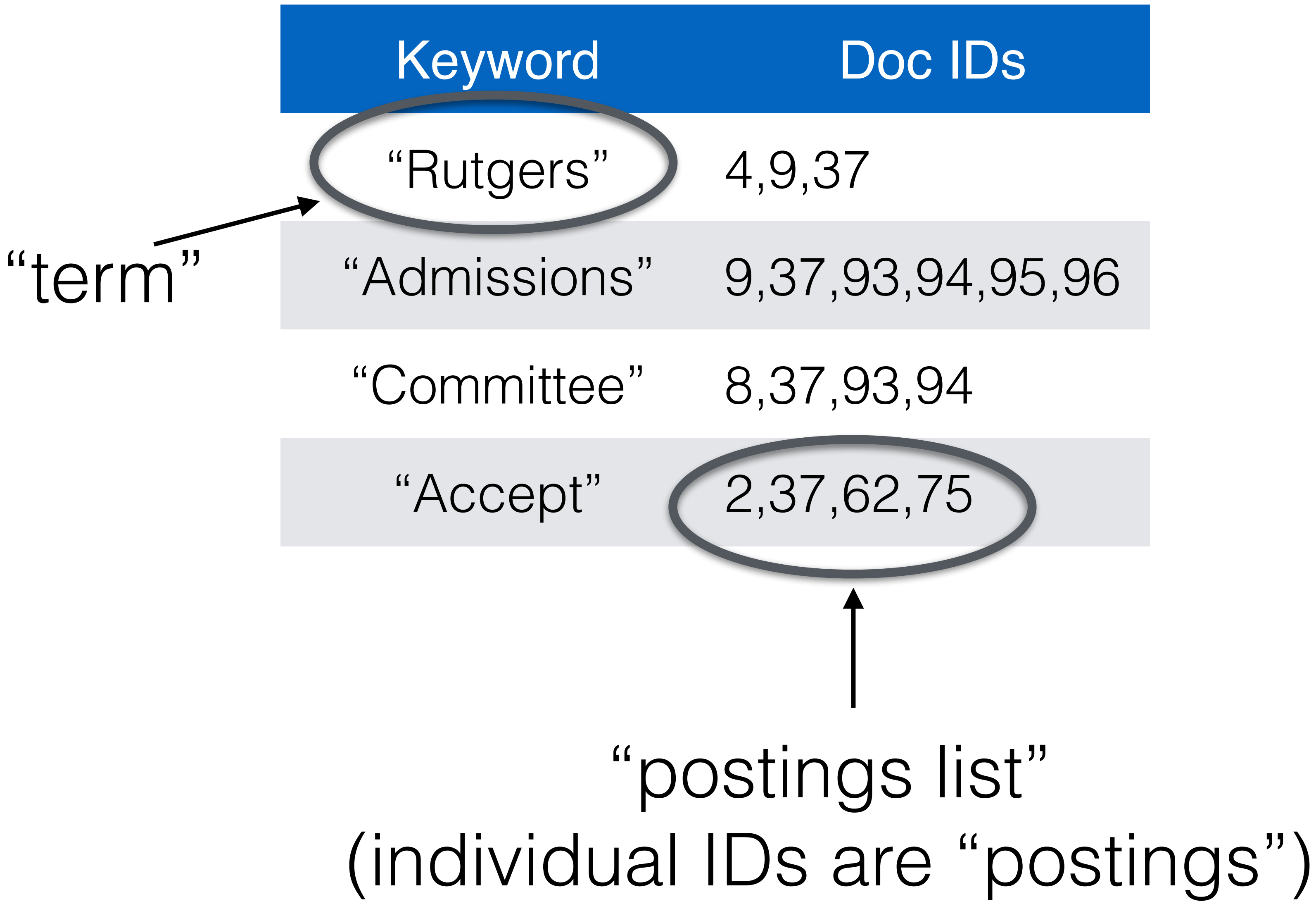
Keyword	Doc IDs
“Rutgers”	4,9,37
“Admissions”	9,37,93,94,95,96
“Committee”	8,37,93,94
“Accept”	2,37,62,75

information retrieval terminology

Keyword	Doc IDs
“Rutgers”	4,9,37
“Admissions”	9,37,93,94,95,96
“Committee”	8,37,93,94
“Accept”	2,37,62,75

“term”

information retrieval terminology



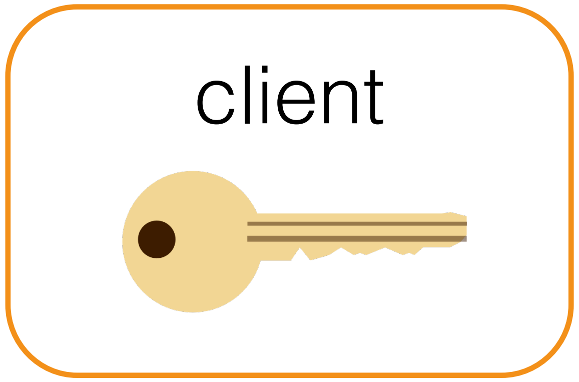
Keyword	Doc IDs
“Rutgers”	4,9,37
“Admissions”	9,37,93,94,95,96
“Committee”	8,37,93,94
“Accept”	2,37,62,75

“term”

“postings list”
(individual IDs are “postings”)

searchable encryption: three algorithms

[Curtmola-Garay-Kamara-Ostrovsky]



term	postings
"Rutgers"	4,9,37
"Admissions"	9,37,93,94,95,96
"Committee"	8,37,93,94
"Accept"	2,37,62,75

cloud

searchable encryption: three algorithms

[Curtmola-Garay-Kamara-Ostrovsky]

client

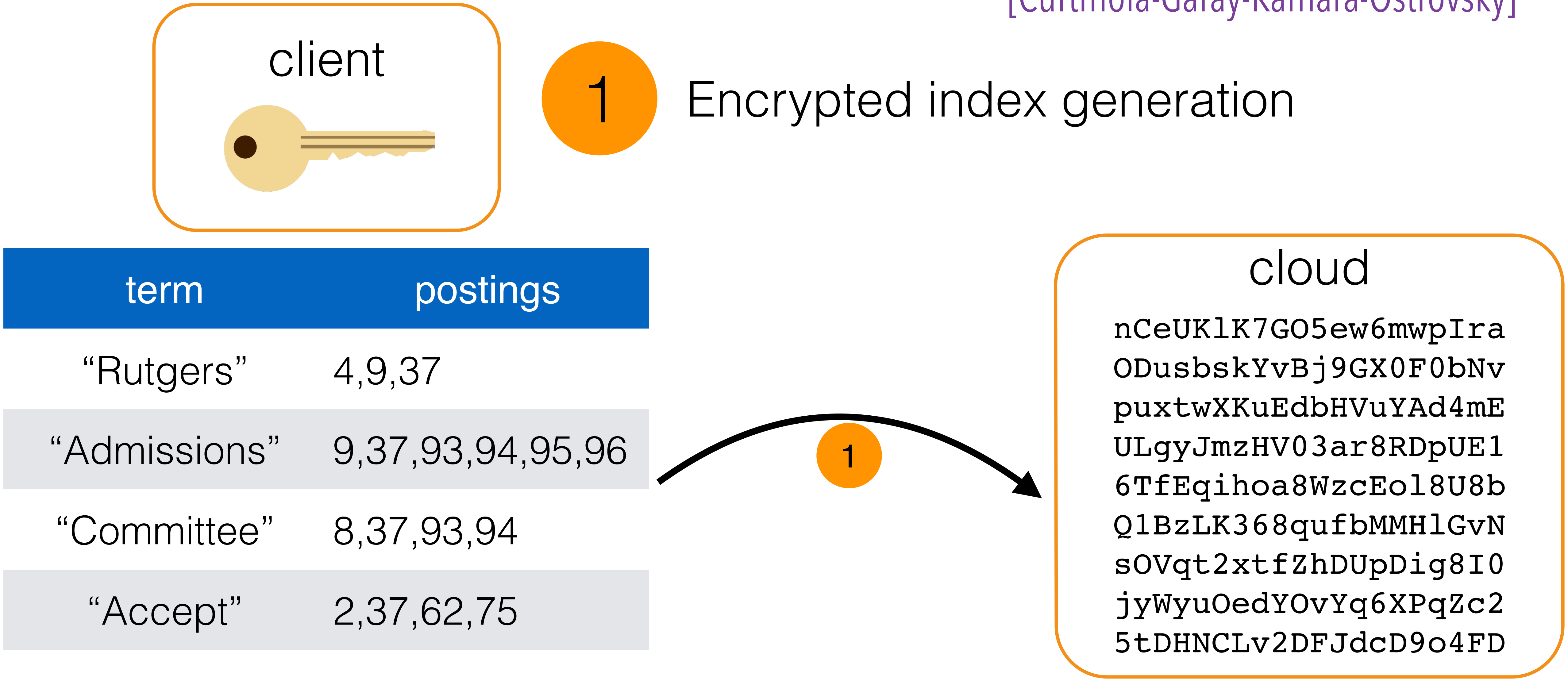
1 Encrypted index generation

term	postings
“Rutgers”	4,9,37
“Admissions”	9,37,93,94,95,96
“Committee”	8,37,93,94
“Accept”	2,37,62,75

cloud

searchable encryption: three algorithms

[Curtmola-Garay-Kamara-Ostrovsky]



1 Encrypted index generation

term	postings
"Rutgers"	4,9,37
"Admissions"	9,37,93,94,95,96
"Committee"	8,37,93,94
"Accept"	2,37,62,75

```
nCeUKlK7GO5ew6mwpIra
ODusbskYvBj9GX0F0bNv
puxtwXKuEdbHVuYAd4mE
ULgyJmzHV03ar8RDpUE1
6TfEqihoa8WzcEol8U8b
Q1BzLK368qufbMMHlGvN
sOVqt2xtfZhDUpDig8I0
jyWyuOedYOvYq6XPqZc2
5tDHNCLv2DFJdcD9o4FD
```

searchable encryption: three algorithms

[Curtmola-Garay-Kamara-Ostrovsky]

client

1 Encrypted index generation

term	postings
“Rutgers”	4,9,37
“Admissions”	9,37,93,94,95,96
“Committee”	8,37,93,94
“Accept”	2,37,62,75

cloud

```
nCeUKlK7GO5ew6mwpIra
ODusbskYvBj9GX0F0bNv
puxtwXKuEdbHVuYAd4mE
ULgyJmzHV03ar8RDpUE1
6TfEqihoa8WzcEol8U8b
Q1BzLK368qufbMMHlGvN
sOVqt2xtfZhDUpDig8I0
jyWyuOedYOvYq6XPqZc2
5tDHNCLv2DFJdcD9o4FD
```

searchable encryption: three algorithms

[Curtmola-Garay-Kamara-Ostrovsky]

client

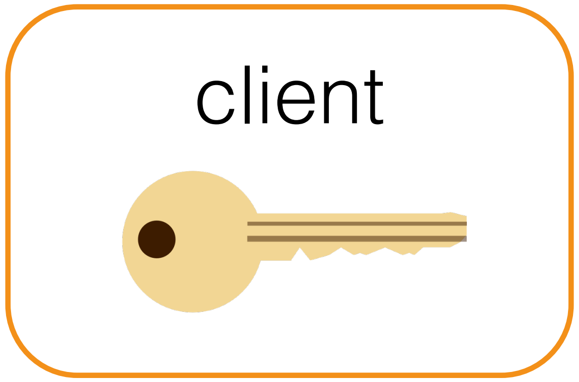
1 Encrypted index generation

cloud

```
nCeUKlK7GO5ew6mwpIra
ODusbskYvBj9GX0F0bNv
puxtwXKuEdbHVuYAd4mE
ULgyJmzHV03ar8RDpUE1
6TfEqihoa8WzcEol8U8b
Q1BzLK368qufbMMHlGvN
sOVqt2xtfZhDUpDig8I0
jyWyuOedYOvYq6XPqZc2
5tDHNCLv2DFJdcD9o4FD
```

searchable encryption: three algorithms

[Curtmola-Garay-Kamara-Ostrovsky]



1 Encrypted index generation

cloud

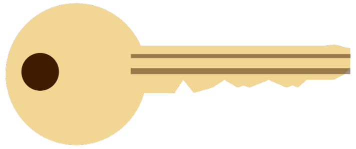
```
nCeUKlK7GO5ew6mwpIra
ODusbskYvBj9GX0F0bNv
puxtwXKuEdbHVuYAd4mE
ULgyJmzHV03ar8RDpUE1
6TfEqihoa8WzcEol8U8b
Q1BzLK368qufbMMHlGvN
sOVqt2xtfZhDUpDig8I0
jyWyu0edYOvYq6XPqZc2
5tDHNCLv2DFJdcD9o4FD
```

2 Token generation

searchable encryption: three algorithms

[Curtmola-Garay-Kamara-Ostrovsky]

client



1 Encrypted index generation

cloud

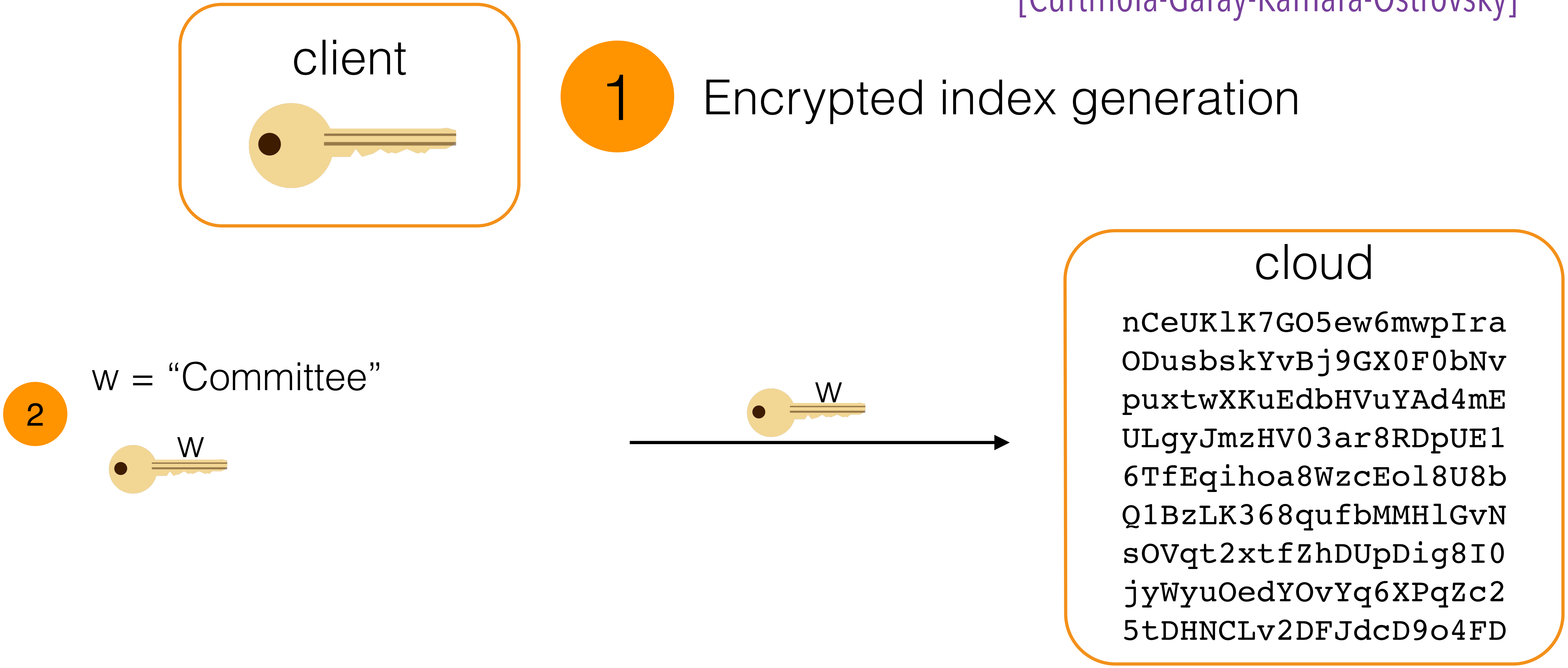
```
nCeUKlK7GO5ew6mwpIra
ODusbskYvBj9GX0F0bNv
puxtwXKuEdbHVuYAd4mE
ULgyJmzHV03ar8RDpUE1
6TfEqihoa8WzcEol8U8b
QlBzLK368qufbMMHlGvN
sOVqt2xtfZhDUpDig8I0
jyWyuOedYOvYq6XPqZc2
5tDHNCLv2DFJdcD9o4FD
```

2 w = “Committee”

2 Token generation

searchable encryption: three algorithms

[Curtmola-Garay-Kamara-Ostrovsky]



client

1 Encrypted index generation

2 w = “Committee”

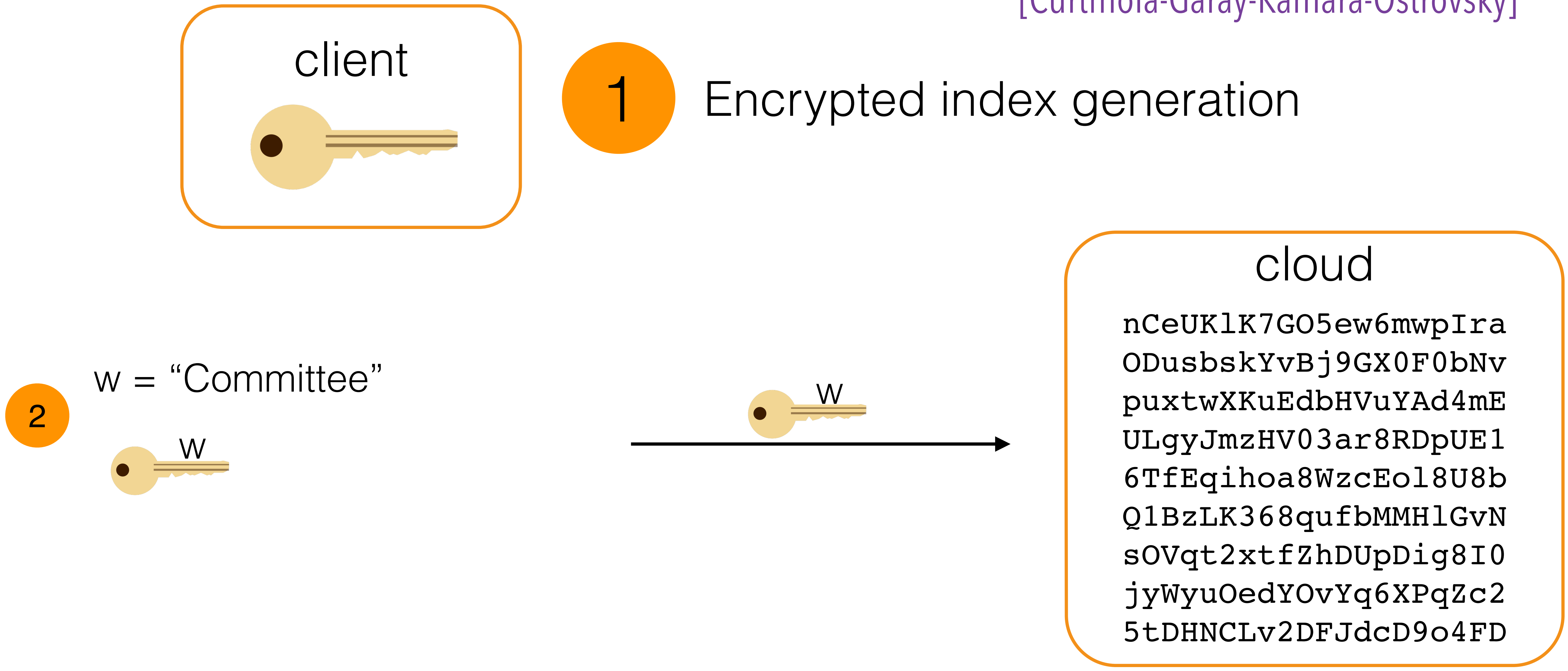
cloud

```
nCeUKlK7GO5ew6mwpIra
ODusbskYvBj9GX0F0bNv
puxtwXKuEdbHVuYAd4mE
ULgyJmzHV03ar8RDpUE1
6TfEqihoa8WzcEol8U8b
Q1BzLK368qufbMMHlGvN
sOVqt2xtfZhDUpDig8I0
jyWyuOedYOvYq6XPqZc2
5tDHNCLv2DFJdcD9o4FD
```

2 Token generation

searchable encryption: three algorithms

[Curtmola-Garay-Kamara-Ostrovsky]



```
nCeUKlK7GO5ew6mwpIra
ODusbskYvBj9GX0F0bNv
puxtwXKuEdbHVuYAd4mE
ULgyJmzHV03ar8RDpUE1
6TfEqihoa8WzcEol8U8b
Q1BzLK368qufbMMHlGvN
sOVqt2xtfZhDUpDig8I0
jyWyuOedYOvYq6XPqZc2
5tDHNCLv2DFJdcD9o4FD
```

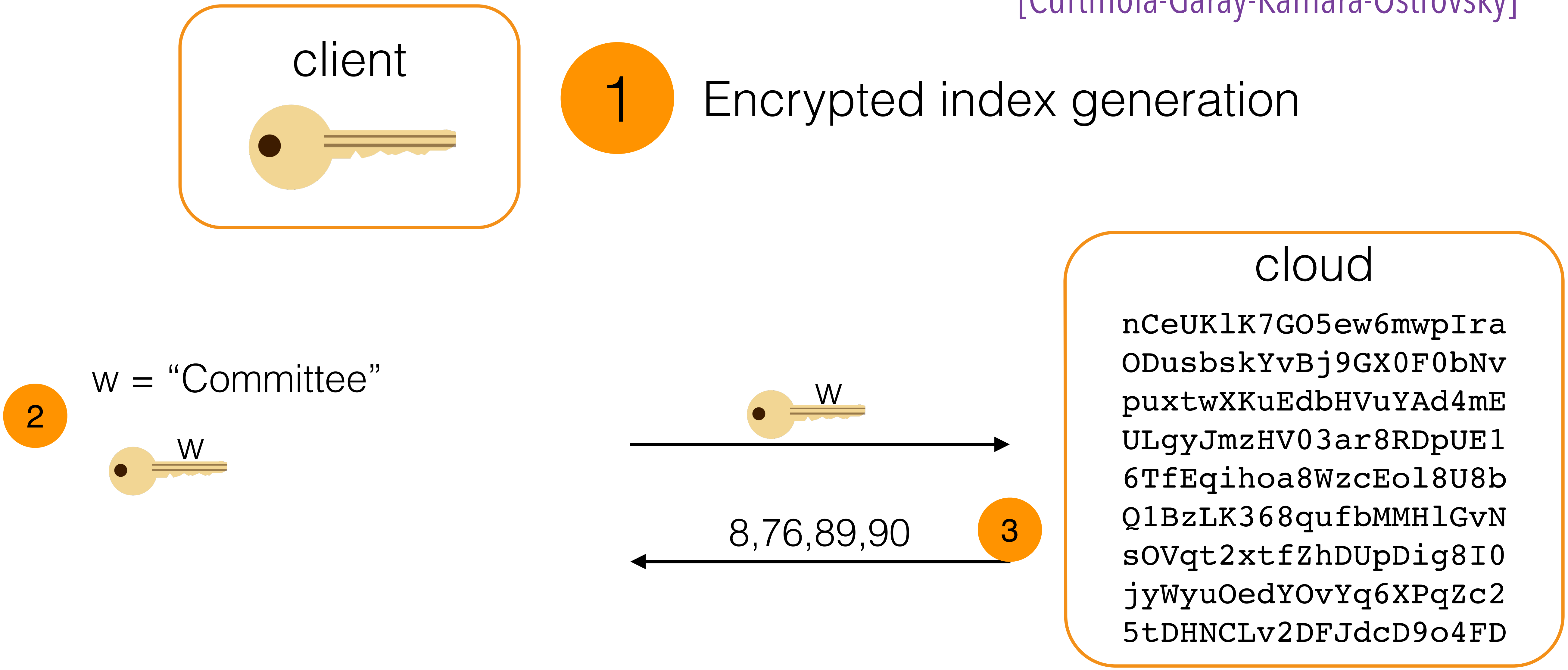
1 Encrypted index generation

2 Token generation

3 Search w/ token

searchable encryption: three algorithms

[Curtmola-Garay-Kamara-Ostrovsky]



client

1 Encrypted index generation

cloud

nCeUKlK7GO5ew6mwpIra
ODusbskYvBj9GX0F0bNv
puxtwXKuEdbHVuYAd4mE
ULgyJmzHV03ar8RDpUE1
6TfEqihoa8WzcEol8U8b
Q1BzLK368qufbMMHlGvN
sOVqt2xtfZhDUpDig8I0
jyWyuOedYOvYq6XPqZc2
5tDHNCLv2DFJdcD9o4FD

2 w = “Committee”

w

w

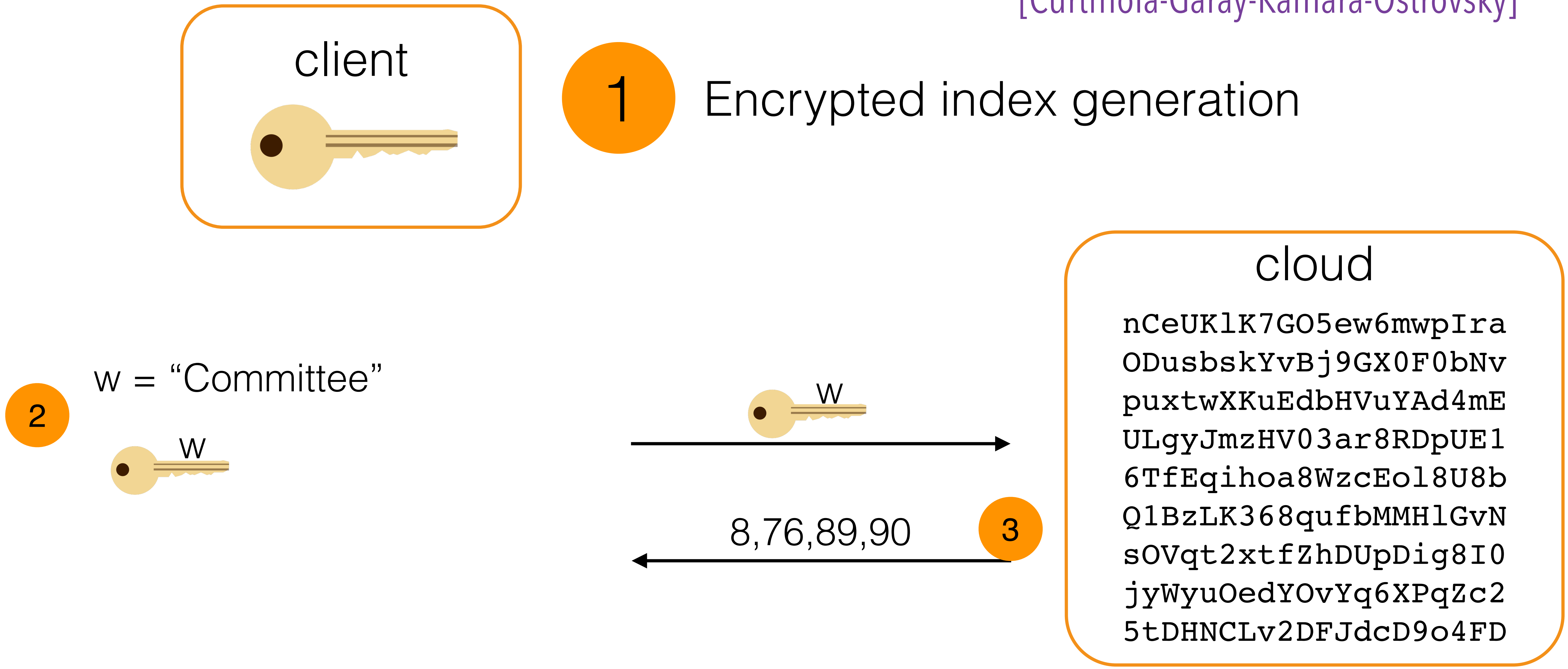
8,76,89,90 3

2 Token generation

3 Search w/ token

searchable encryption: three algorithms

[Curtmola-Garay-Kamara-Ostrovsky]



client

1 Encrypted index generation

2 w = “Committee”

w

w

8,76,89,90

3

cloud

nCeUKlK7GO5ew6mwpIra
ODusbskYvBj9GX0F0bNv
puxtwXKuEdbHVuYAd4mE
ULgyJmzHV03ar8RDpUE1
6TfEqihoa8WzcEol8U8b
QlBzLK368qufbMMHlGvN
sOVqt2xtfZhDUpDig8I0
jyWyuOedYOvYq6XPqZc2
5tDHNCLv2DFJdcD9o4FD

2 Token generation

3 Search w/ token

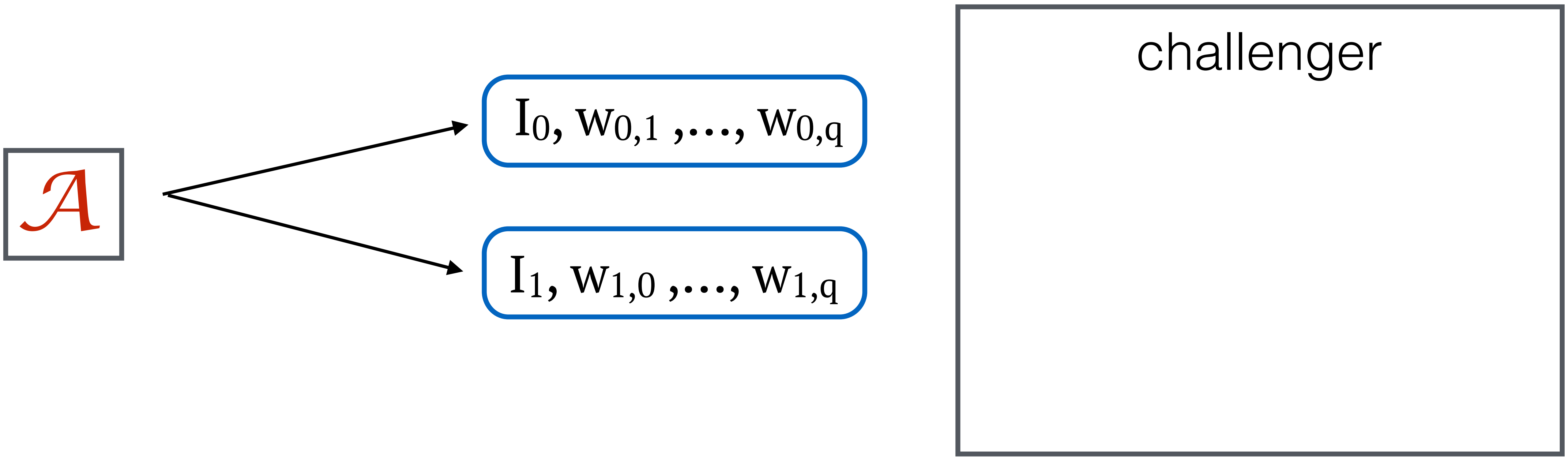
correctness ⇒ server learns postings for w

searchable encryption security definition [CGKO]

$\mathcal{A}$

challenger

searchable encryption security definition [CGKO]



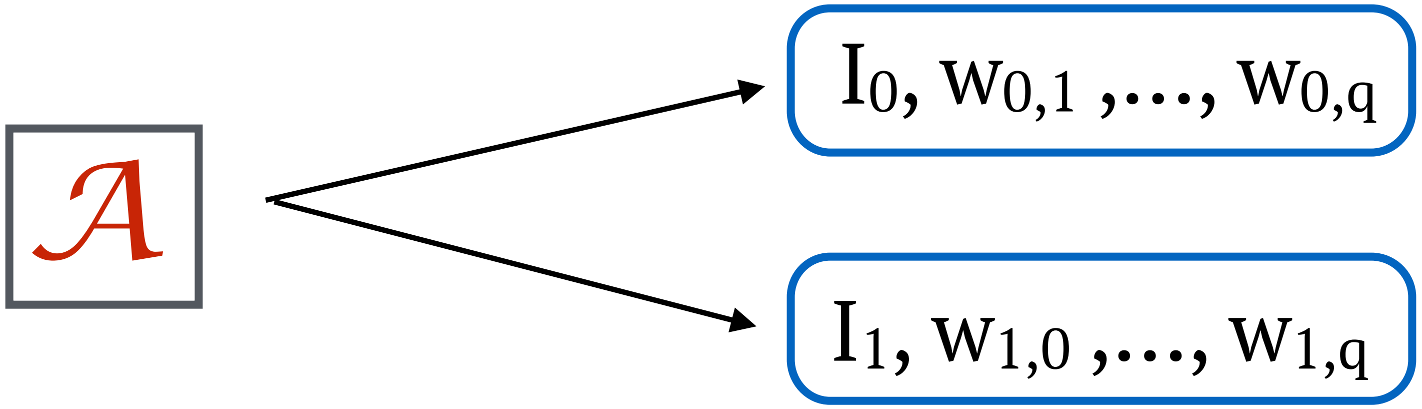
$\mathcal{A}$

$I_0, w_{0,1} ,\ldots, w_{0,q}$

$I_1, w_{1,0} ,\ldots, w_{1,q}$

challenger

searchable encryption security definition [CGKO]



$\mathcal{A}$

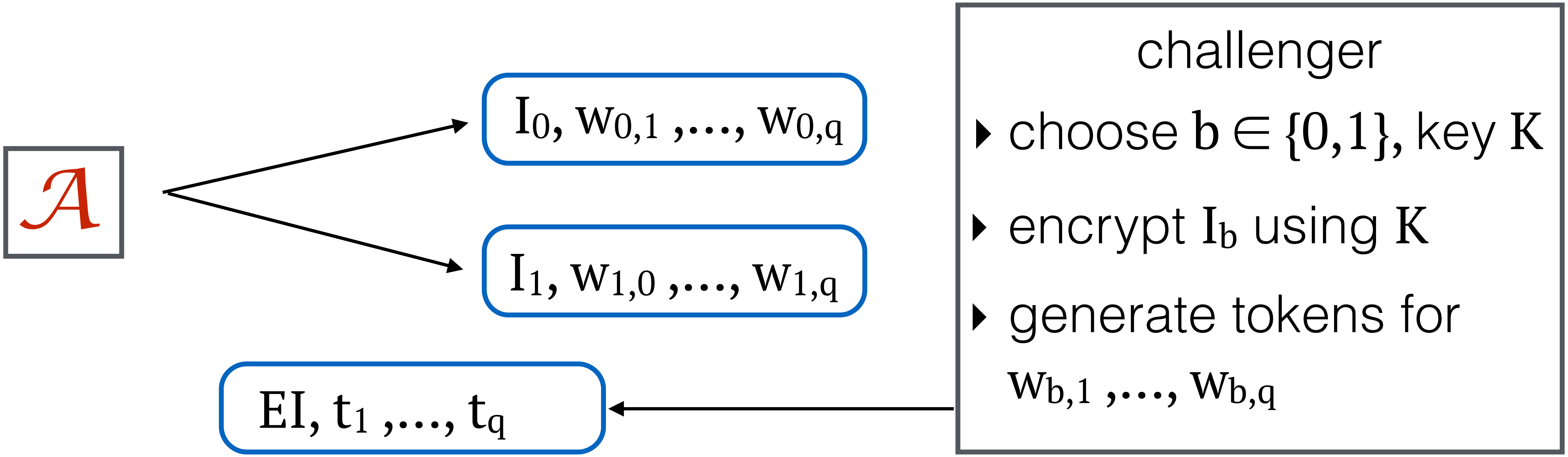
$I_0, w_{0,1}, \ldots, w_{0,q}$

$I_1, w_{1,0}, \ldots, w_{1,q}$

challenger

- choose $b \in \{0,1\}$, key K
- encrypt I_b using K
- generate tokens for $w_{b,1}, \ldots, w_{b,q}$

searchable encryption security definition [CGKO]



$\mathcal{A}$

$I_0, w_{0,1}, \ldots, w_{0,q}$

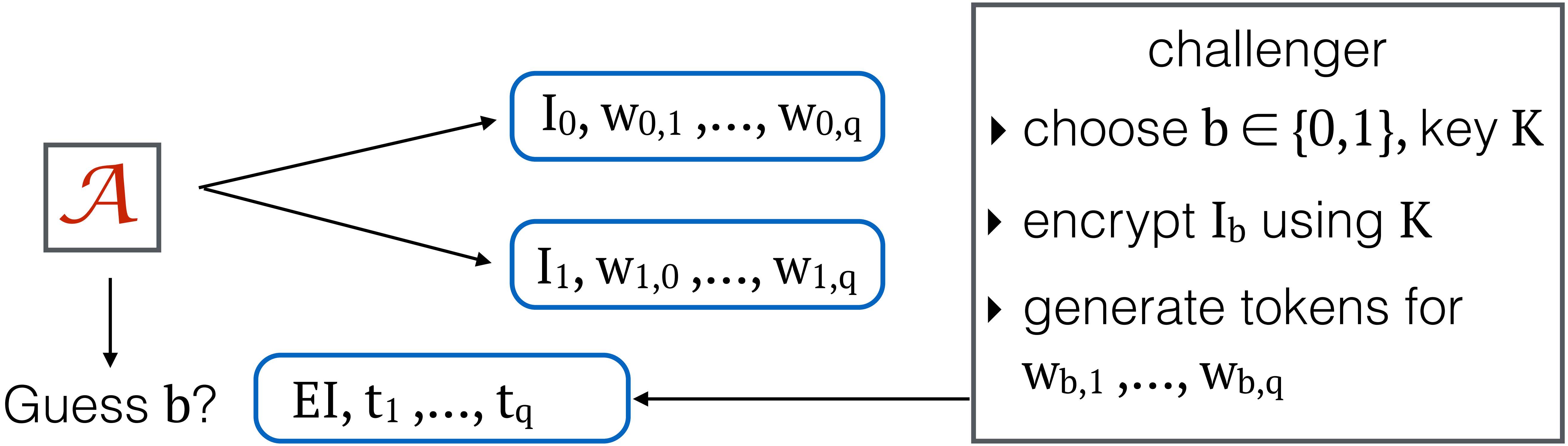
$I_1, w_{1,0}, \ldots, w_{1,q}$

$EI, t_1, \ldots, t_q$

challenger

- choose $b \in \{0,1\}$, key K
- encrypt I_b using K
- generate tokens for $w_{b,1}, \ldots, w_{b,q}$

searchable encryption security definition [CGKO]



$\mathcal{A}$

$I_0, w_{0,1}, \ldots, w_{0,q}$

$I_1, w_{1,0}, \ldots, w_{1,q}$

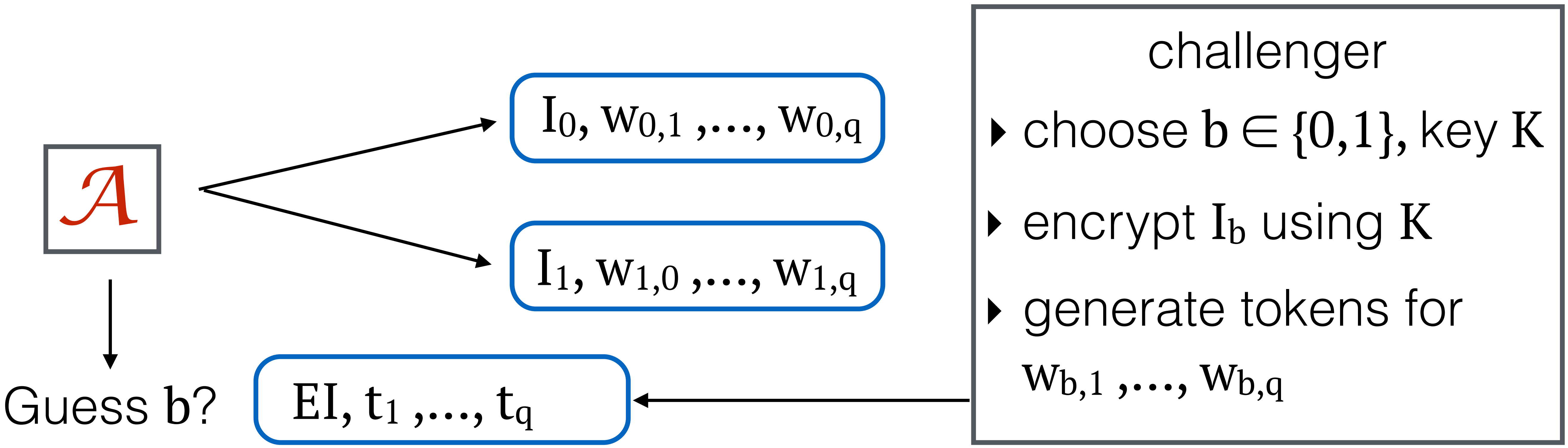
Guess b?

$EI, t_1, \ldots, t_q$

challenger

- choose $b \in \{0,1\}$, key K
- encrypt I_b using K
- generate tokens for $w_{b,1}, \ldots, w_{b,q}$

searchable encryption security definition [CGKO]



$\mathcal{A}$

$I_0, w_{0,1}, \ldots, w_{0,q}$

$I_1, w_{1,0}, \ldots, w_{1,q}$

challenger

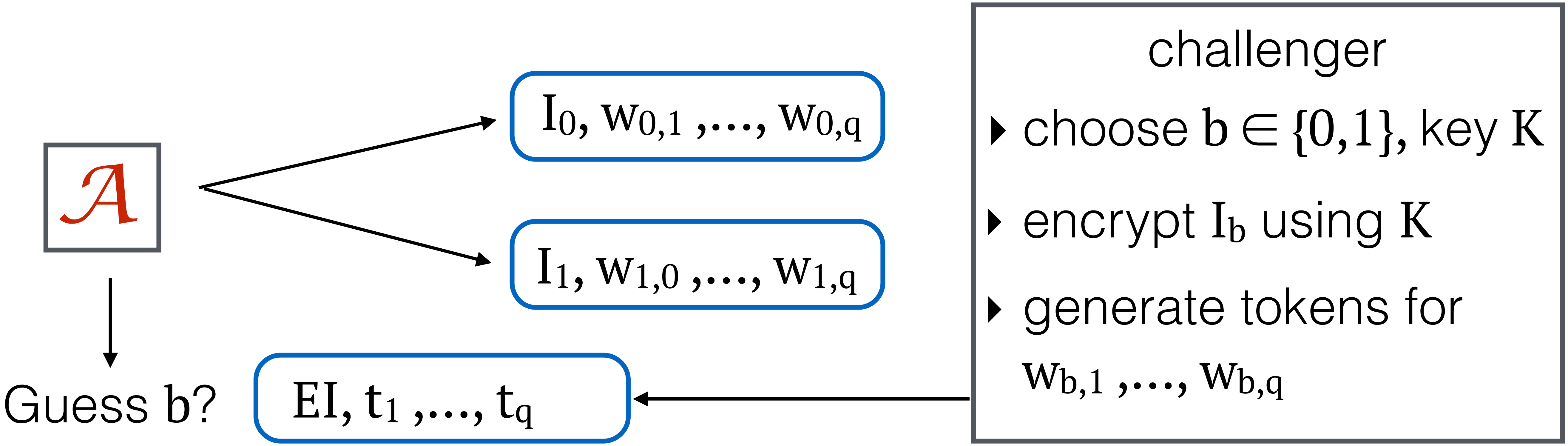
- choose $b \in \{0,1\}$, key K
- encrypt I_b using K
- generate tokens for $w_{b,1}, \ldots, w_{b,q}$

$EI, t_1, \ldots, t_q$

Guess b?

- Restrictions to prevent trivial attacks:
  - I_0, I_1 have same no. postings
  - Same postings list for each $w_{0,i}$ and $w_{1,i}$
    - Notation: $I_0[w_{0,i}] = I_1[w_{1,i}]$

searchable encryption security definition [CGKO]



$\mathcal{A}$

$I_0, w_{0,1}, \ldots, w_{0,q}$

$I_1, w_{1,0}, \ldots, w_{1,q}$

Guess b?

$EI, t_1, \ldots, t_q$

challenger

- choose $b \in \{0,1\}$, key K
- encrypt I_b using K
- generate tokens for $w_{b,1}, \ldots, w_{b,q}$

- Restrictions to prevent trivial attacks:
  - I_0, I_1 have same no. postings
  - Same postings list for each $w_{0,i}$ and $w_{1,i}$
    - Notation: $I_0[w_{0,i}] = I_1[w_{1,i}]$

__Def__: Scheme is __**secure**__ if all poly-time $\mathcal{A}$ guess b with probability $\approx 1/2$

what does searchable encryption leak?

will not hide:

- postings lists as searches are issued
- when searches repeat
- total # postings in index

hides everything for part of index not searched, including:

- sizes of postings lists
- postings in lists
- # of postings lists in index

research on searchable encryption

- secure updates after initial upload [KPR, KP, CJJJKRS, NPG]
- other security properties (auth, UC, etc) [KO, LSDHJ, CK]
- boolean search queries [CJJKRS]
- keyword search with “web structure” [CK]
- used in DB encryption in CryptDB & Monomi [PRZB, TKMZ]
- Challenges with flexibility, usability

bottleneck of searchable encryption: locality

systems collaborators at IBM complained:

> “ Fine, the asymptotics are optimal, but this stuff is unusably slow for large indexes.

bottleneck of searchable encryption: locality

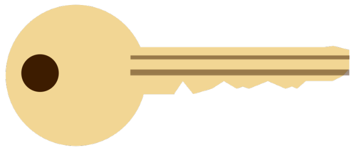
systems collaborators at IBM complained:

> “ Fine, the asymptotics are optimal, but this stuff is unusably slow for large indexes.

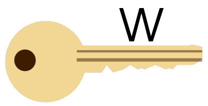
➡ Runtime bottleneck: disk latency, not crypto processing.

memory access during encrypted search

client



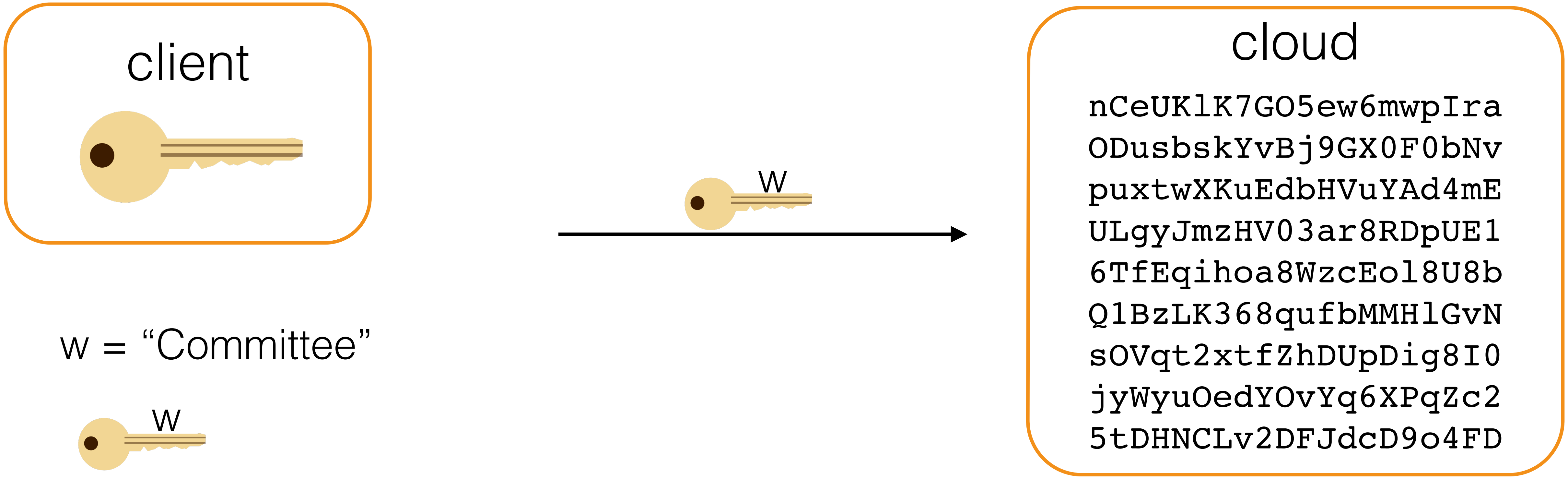
w = “Committee”



cloud

nCeUKlK7GO5ew6mwpIra
ODusbskYvBj9GX0F0bNv
puxtwXKuEdbHVuYAd4mE
ULgyJmzHV03ar8RDpUE1
6TfEqihoa8WzcEol8U8b
Q1BzLK368qufbMMHlGvN
sOVqt2xtfZhDUpDig8I0
jyWyuOedYOvYq6XPqZc2
5tDHNCLv2DFJdcD9o4FD

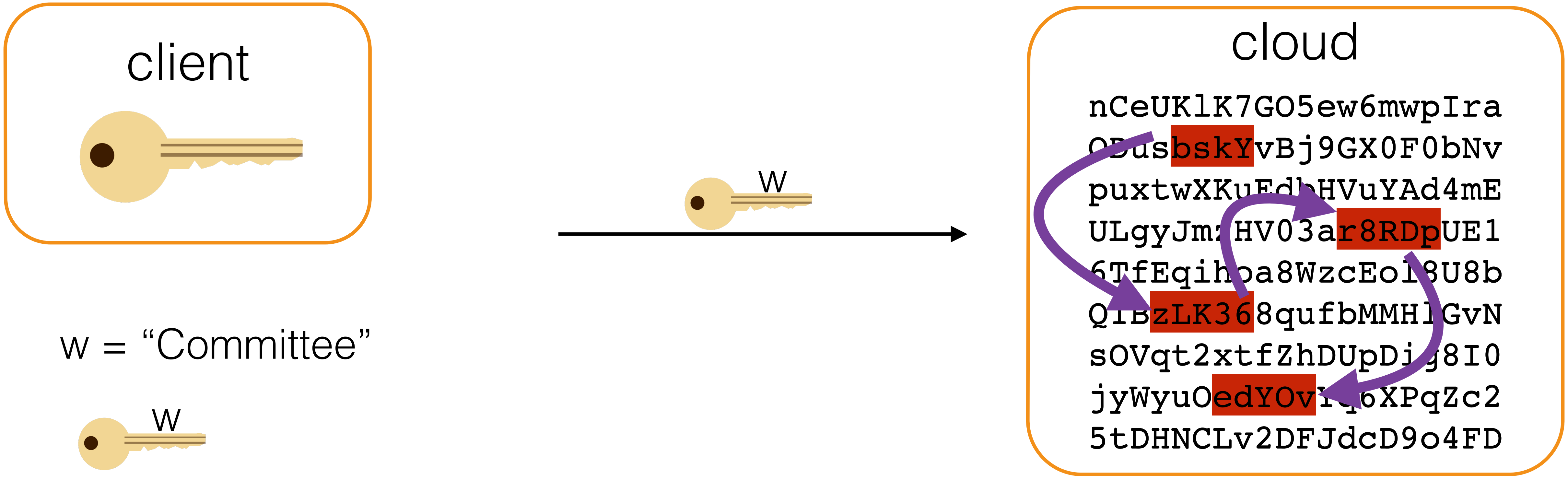
memory access during encrypted search



w = “Committee”

```
nCeUKlK7GO5ew6mwpIra
ODusbskYvBj9GX0F0bNv
puxtwXKuEdbHVuYAd4mE
ULgyJmzHV03ar8RDpUE1
6TfEqihoa8WzcEol8U8b
Q1BzLK368qufbMMHlGvN
sOVqt2xtfZhDUpDig8I0
jyWyu0edYOvYq6XPqZc2
5tDHNCLv2DFJdcD9o4FD
```

memory access during encrypted search



client

w

w = "Committee"

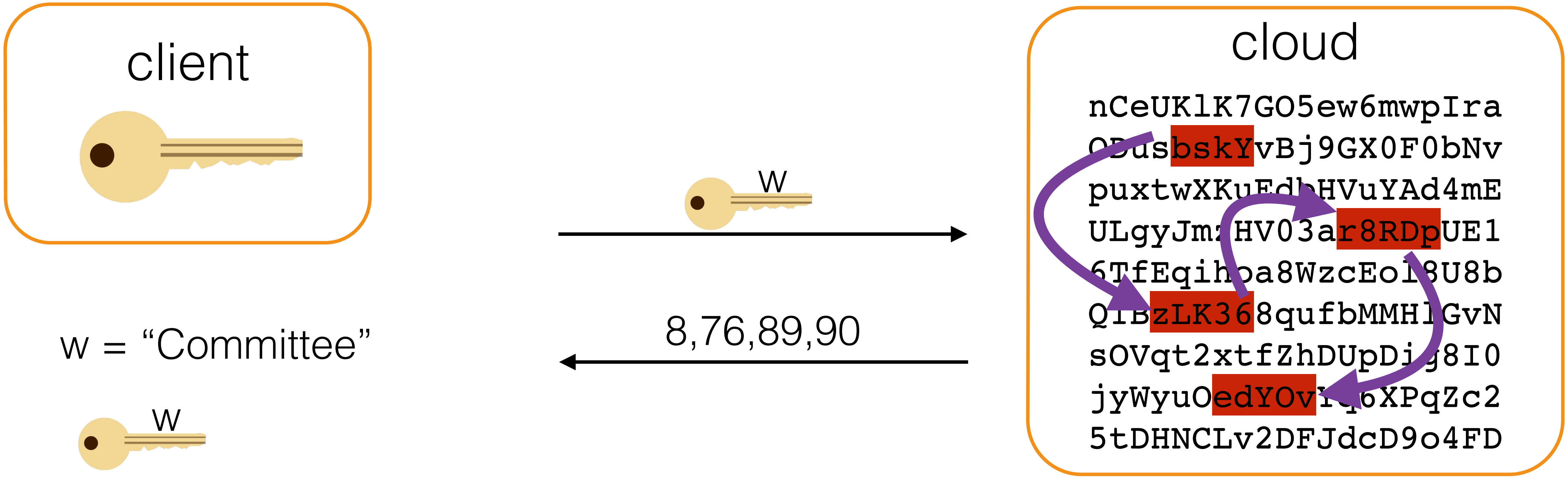
w

cloud

nCeUKlK7GO5ew6mwpIra
QDusbskYvBj9GX0F0bNv
puxtwXKuFdbHVuYAd4mE
ULgyJmzHV03ar8RDpUE1
6TfEqihpa8WzcEoI8U8b
Q1BzLK368qufbMMHlGvN
sOVqt2xtfZhDUpDij8I0
jyWyuOedYOvYq6XPqZc2
5tDHNCLv2DFJdcD9o4FD

➡ constructions access one random part of memory per posting

memory access during encrypted search



client

w

8,76,89,90

w = “Committee”

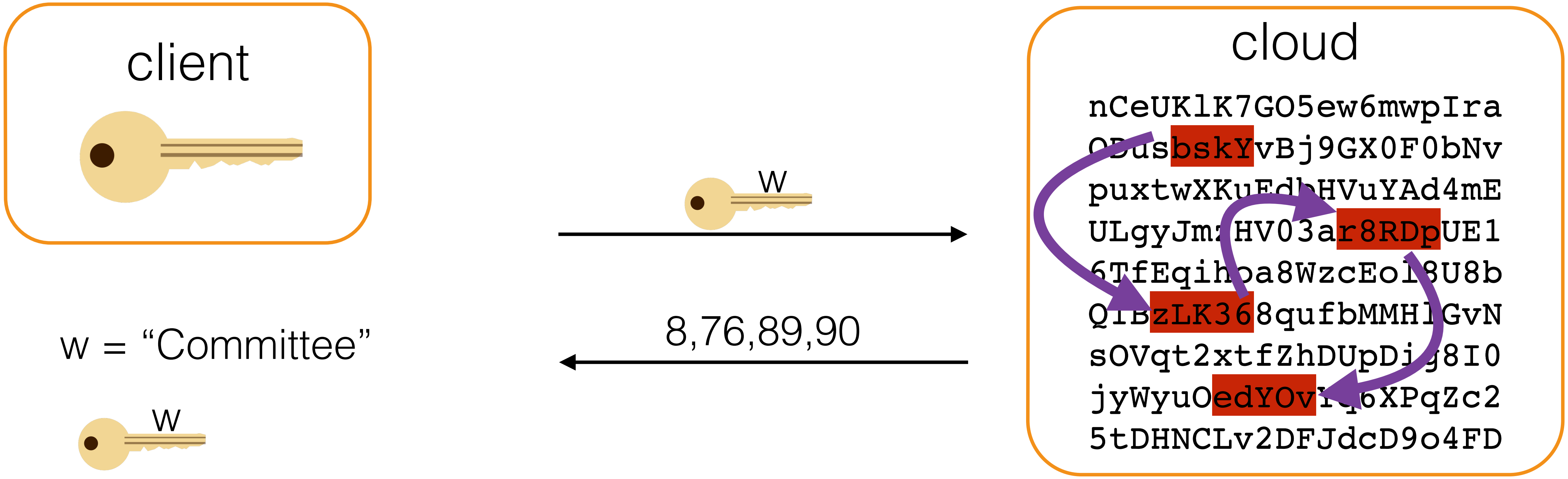
w

cloud

nCeUKlK7GO5ew6mwpIra
ODusbskYvBj9GX0F0bNv
puxtwXKuFdbHVuYAd4mE
ULgyJmzHV03ar8RDpUE1
6TfEqihpa8WzcEol8U8b
QIBzLK368qufbMMHlGvN
sOVqt2xtfZhDUpDij8I0
jyWyuOedYOvYq6XPqZc2
5tDHNCLv2DFJdcD9o4FD

➡ constructions access one random part of memory per posting

memory access during encrypted search



client

w = “Committee”

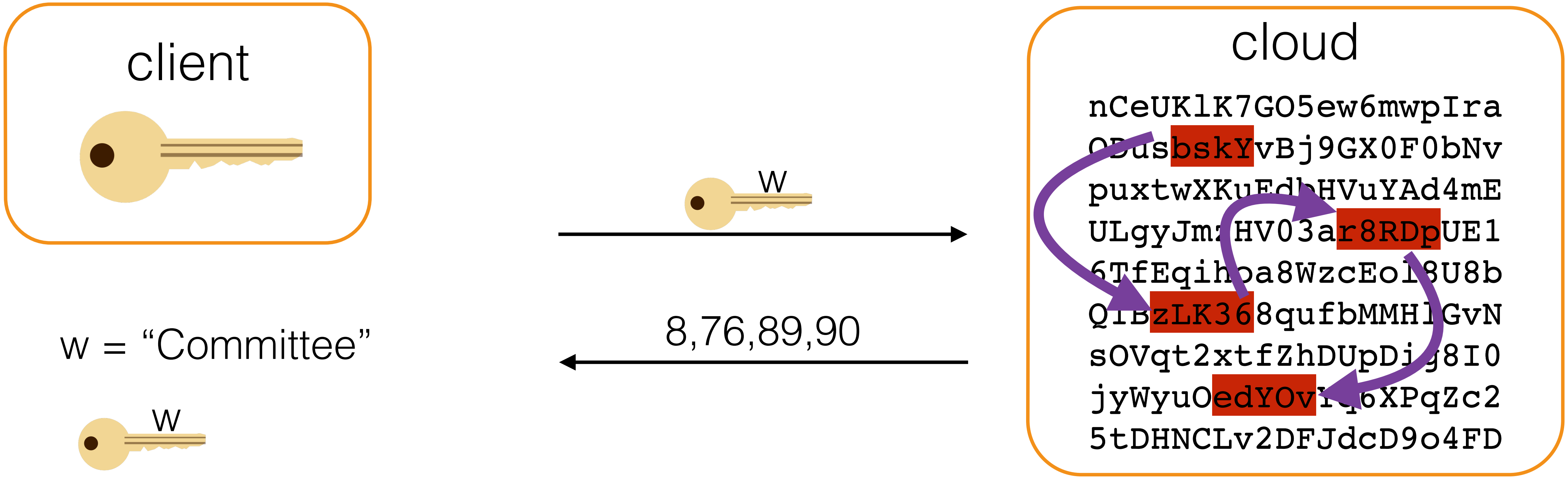
cloud

```
nCeUKlK7GO5ew6mwpIra
QDusbskYvBj9GX0F0bNv
puxtwXKuEdbHVuYAd4mE
ULgyJmzHV03ar8RDpUE1
6TfEqihoa8WzcEol8U8b
QIBzLK368qufbMMHlGvN
sOVqt2xtfZhDUpDig8I0
jyWyuOedYOvYq6XPqZc2
5tDHNCLv2DFJdcD9o4FD
```

➡ constructions access one random part of memory per posting

- one disk seek per posting (≈ only few bytes, wasteful)

memory access during encrypted search



client

w = “Committee”

w

w

8,76,89,90

cloud

nCeUKlK7GO5ew6mwpIra
ODusbskYvBj9GX0F0bNv
puxtwXKuFdbHVuYAd4mE
ULgyJmzHV03ar8RDpUE1
6TfEqihpa8WzcEol8U8b
QIBzLK368qufbMMHlGvN
sOVqt2xtfZhDUpDig8I0
jyWyuOedYOvYq6XPqZc2
5tDHNCLv2DFJdcD9o4FD

➡ constructions access one random part of memory per posting

- one disk seek per posting (≈ only few bytes, wasteful)

➡ plaintext search can use one contiguous access for entire postings list

i/o efficiency theory

- count *only* # of blocks moved to/from disk [Aggarwal-Vitter]
  - to read a block in new location, incur seek time
  - seek time overwhelms time for computation
- numerous versions of theory i/o models (see [Vitter] text)
  - optimal results (matching upper/lower bounds) for many problems like sorting, dictionary look-up, …

our results: i/o efficiency and searchable encryption

- ➡initiate study of i/o efficiency and security
  - ‣ first formal connection with any crypto primitive

our results: i/o efficiency and searchable encryption

- ➡initiate study of i/o efficiency and security
  - ‣ first formal connection with any crypto primitive
- ➡unconditional i/o lower bounds for searchable encryption
  - ‣ new proof technique

our results: i/o efficiency and searchable encryption

- ➡initiate study of i/o efficiency and security
  - ‣ first formal connection with any crypto primitive
- ➡unconditional i/o lower bounds for searchable encryption
  - ‣ new proof technique
- ➡construction improving i/o efficiency of prior work

our results: i/o efficiency lower bound

“Theorem”: Secure searchable encryption must either:

(1) Have a very large encrypted index,

or

(2) Read memory in a highly “non-local” fashion,

or

(3) Read more memory than a plaintext search.

our results: i/o efficiency lower bound

"Theorem": Secure searchable encryption must either:

(1) Have a very large encrypted index,

or

(2) Read memory in a highly "non-local" fashion,

or

(3) Read more memory than a plaintext search.

➡ unconditional (no complexity assumptions)

our results: i/o efficiency lower bound

"Theorem": Secure searchable encryption must either:

(1) Have a very large encrypted index,

or

(2) Read memory in a highly "non-local" fashion,

or

(3) Read more memory than a plaintext search.

- ➡ unconditional (no complexity assumptions)
- ➡ applies to any scheme (no assumption about how it works)

our results: i/o efficiency lower bound

"Theorem": Secure searchable encryption must either:

(1) Have a very large encrypted index,

or

(2) Read memory in a highly "non-local" fashion,

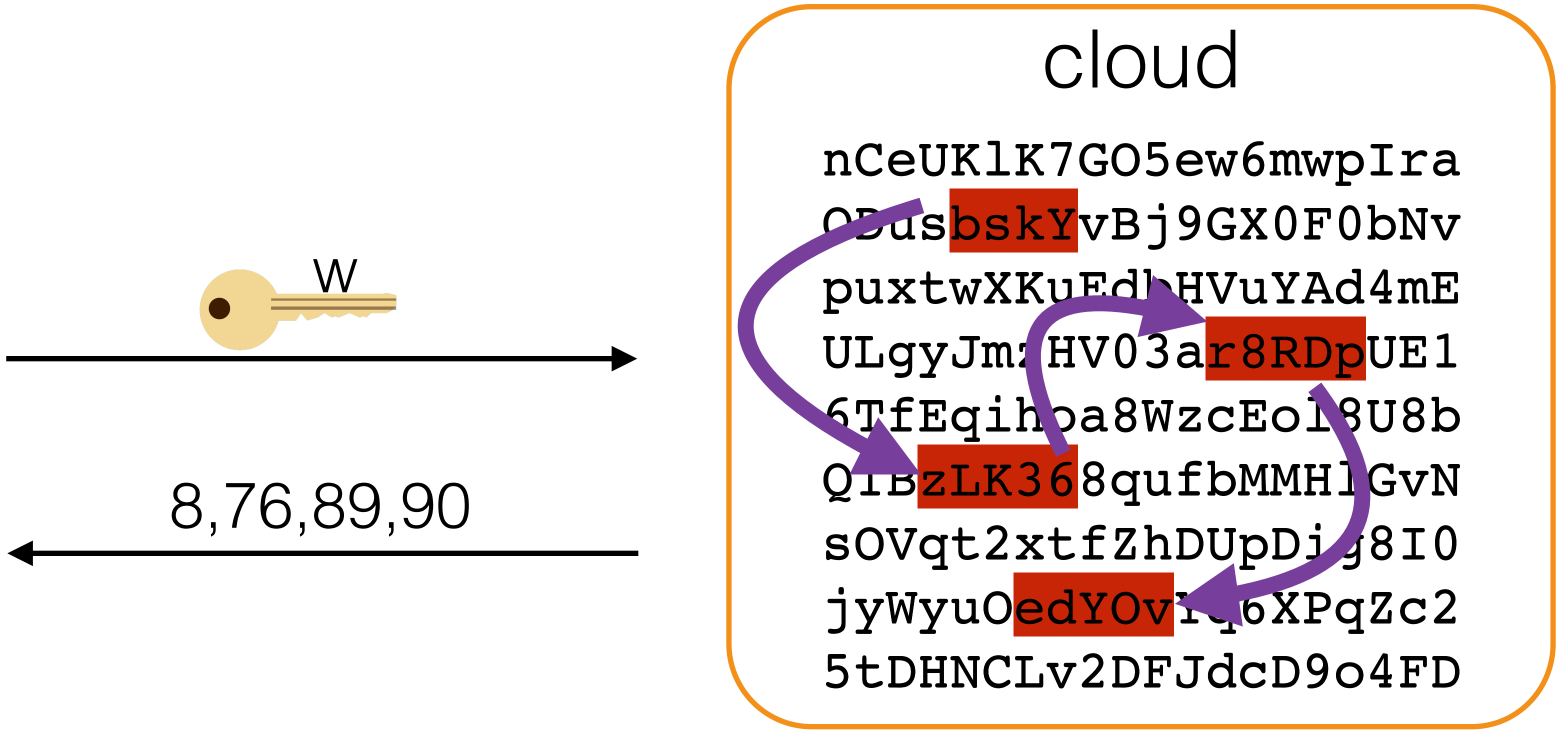
or

(3) Read more memory than a plaintext search.

- ➡ unconditional (no complexity assumptions)
- ➡ applies to any scheme (no assumption about how it works)
- ➡ different type of i/o lower bound: security vs. correctness

memory utilization in searching

any construction can be seen as “touching” contiguous regions of memory during search processing:



```
nCeUKlK7GO5ew6mwpIra
QDusbskYvBj9GX0F0bNv
puxtwXKuEdbHVuYAd4mE
ULgyJmzHV03ar8RDpUE1
6TfEqihpa8WzcEol8U8b
QIBzLK368qufbMMHlGvN
sOVqt2xtfZhDUpDig8I0
jyWyuOedYOvYq6XPqZc2
5tDHNCLv2DFJdcD9o4FD
```

memory utilization in searching

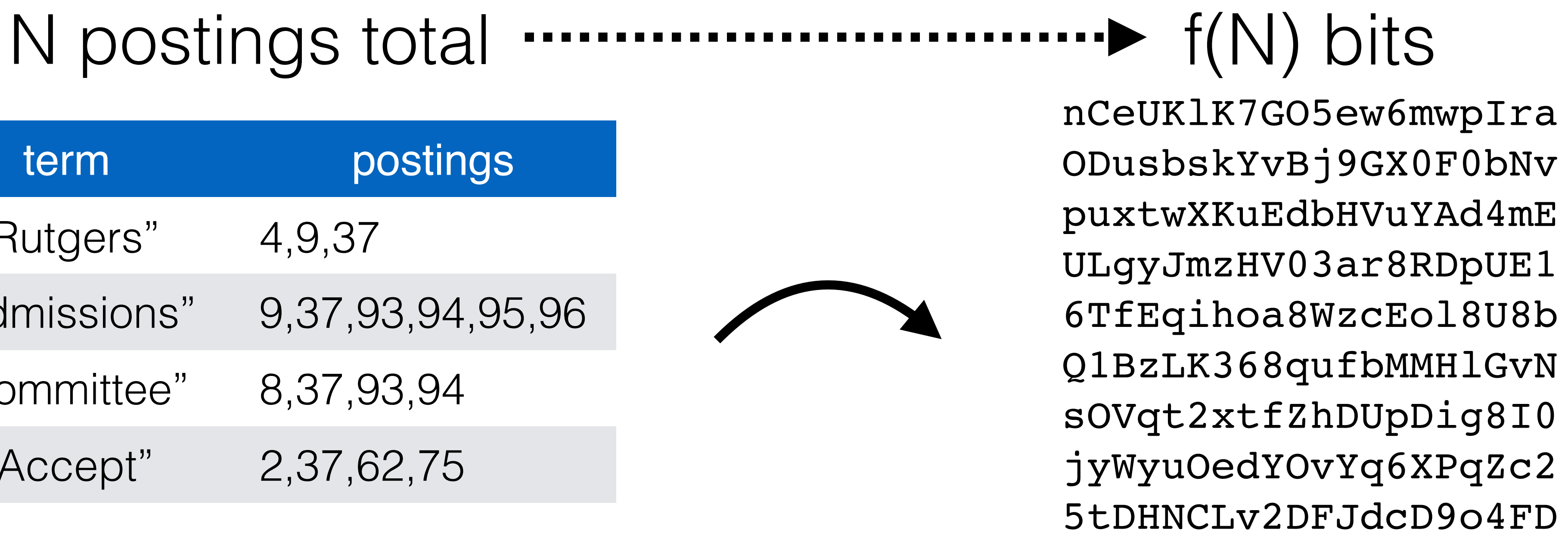
we use three (very primitive) measures:

1. encrypted index size: measured relative to #-postings

memory utilization in searching

we use three (very primitive) measures:

1. encrypted index size: measured relative to #-postings



N postings total

f(N) bits

term	postings
“Rutgers”	4,9,37
“Admissions”	9,37,93,94,95,96
“Committee”	8,37,93,94
“Accept”	2,37,62,75

```
nCeUKlK7GO5ew6mwpIra
ODusbskYvBj9GX0F0bNv
puxtwXKuEdbHVuYAd4mE
ULgyJmzHV03ar8RDpUE1
6TfEqihoa8WzcEol8U8b
Q1BzLK368qufbMMHlGvN
sOVqt2xtfZhDUpDig8I0
jyWyuOedYOvYq6XPqZc2
5tDHNCLv2DFJdcD9o4FD
```

memory utilization in searching

we use three (very primitive) measures:

1. encrypted index size: measured relative to #-postings

memory utilization in searching

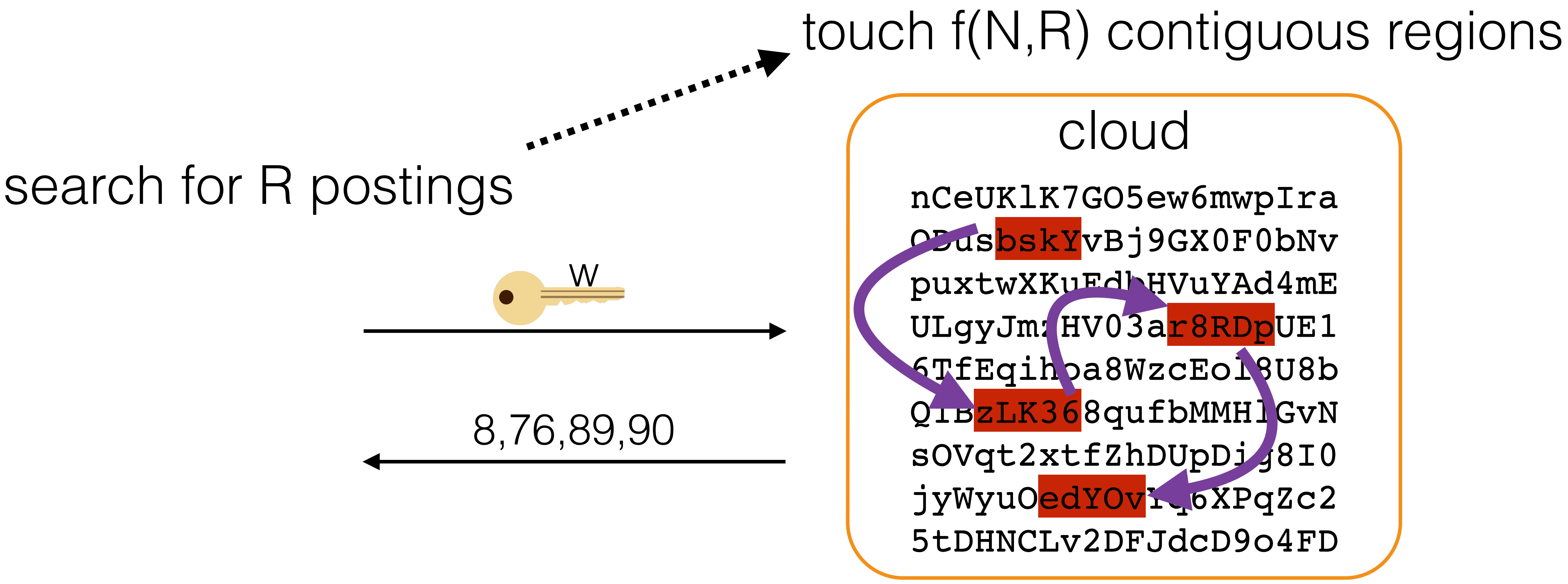
we use three (very primitive) measures:

1. encrypted index size: measured relative to #-postings
2. locality: number of contiguous regions touched

memory utilization in searching

we use three (very primitive) measures:

1. encrypted index size: measured relative to #-postings
2. locality: number of contiguous regions touched



memory utilization in searching

we use three (very primitive) measures:

1. encrypted index size: measured relative to #-postings
2. locality: number of contiguous regions touched
3. read overlaps: amount of touched memory common between searches

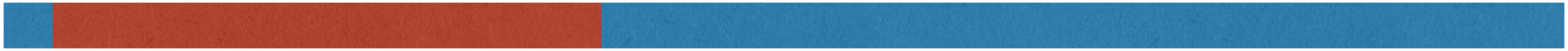
read overlaps

Encrypted index in memory:

read overlaps

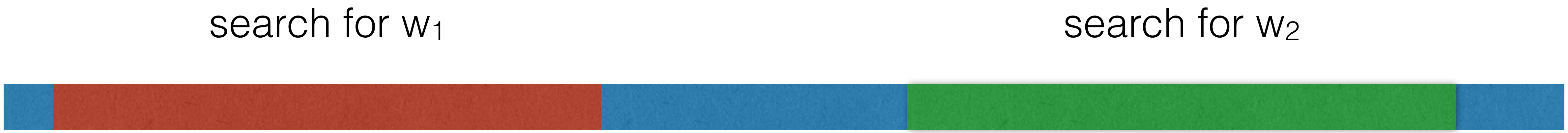
Encrypted index in memory:

search for w_1



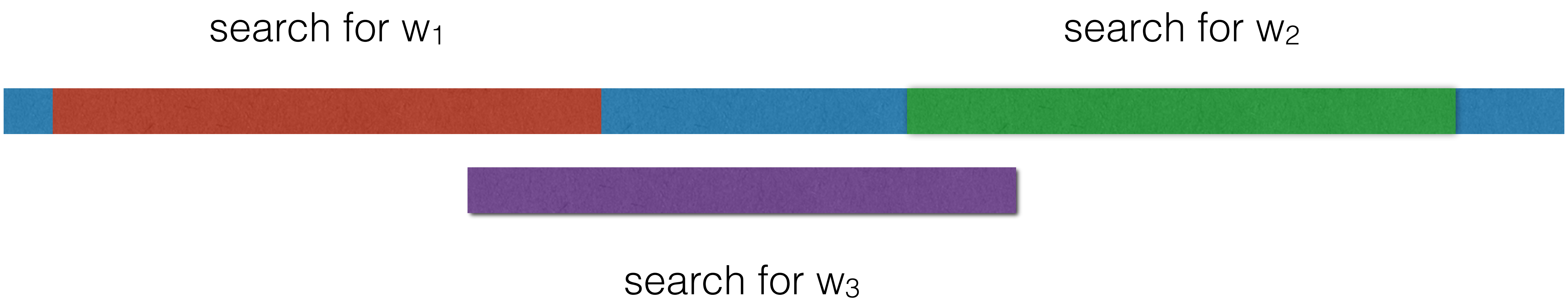
read overlaps

Encrypted index in memory:



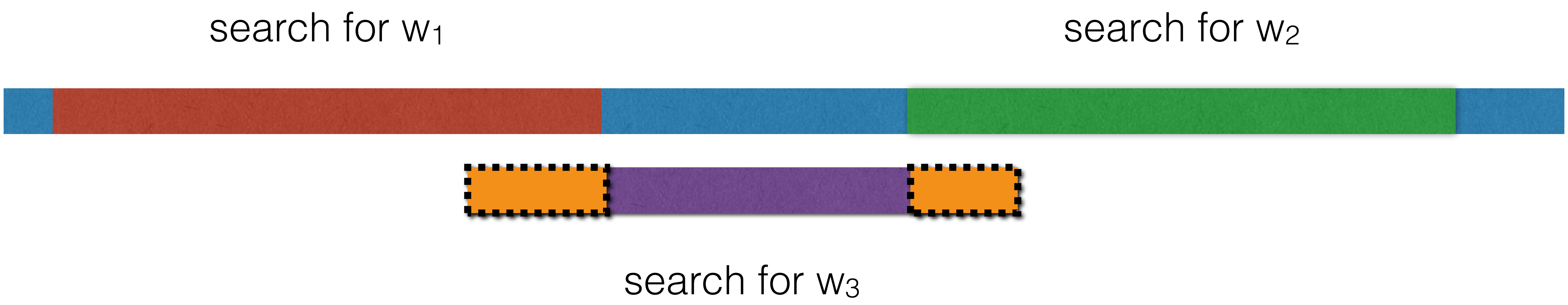
read overlaps

Encrypted index in memory:



read overlaps

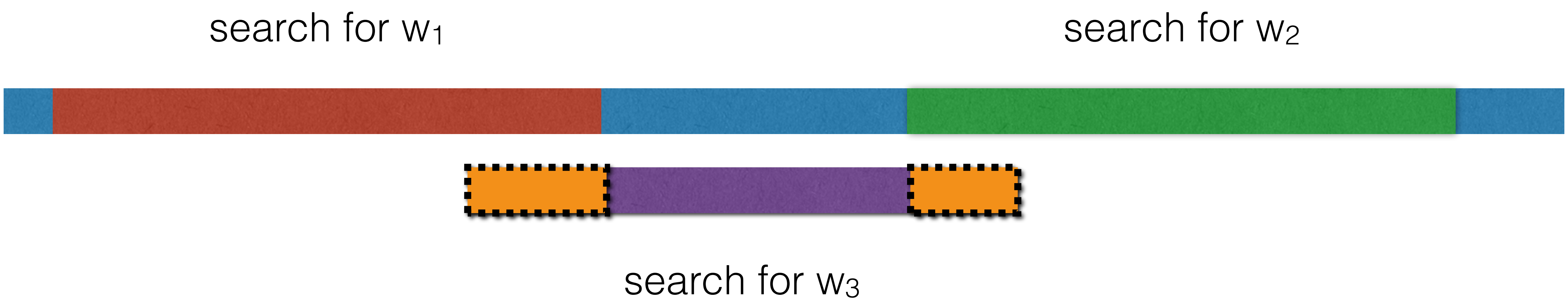
Encrypted index in memory:



Overlap of search for w_3 = size of orange regions

read overlaps

Encrypted index in memory:

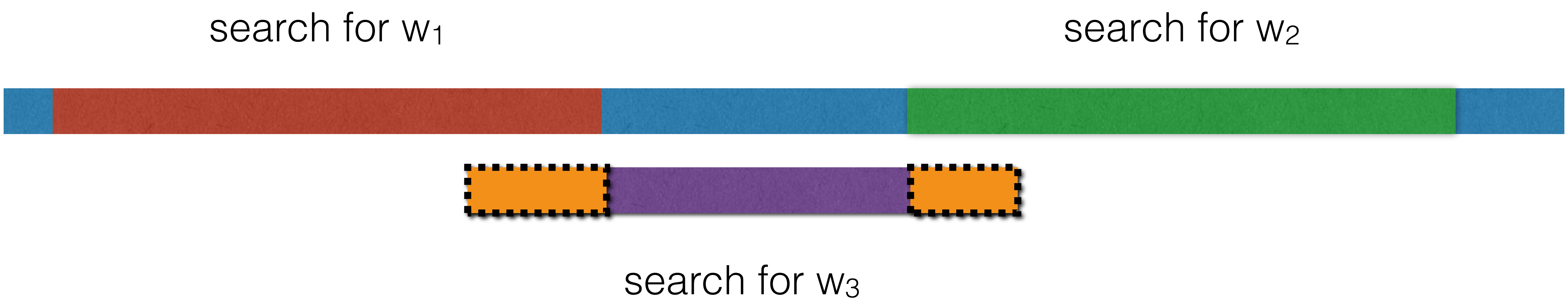


Overlap of search for w_3 = size of orange regions

➡ f-overlap $\Longrightarrow$ any search touches f common bits

read overlaps

Encrypted index in memory:

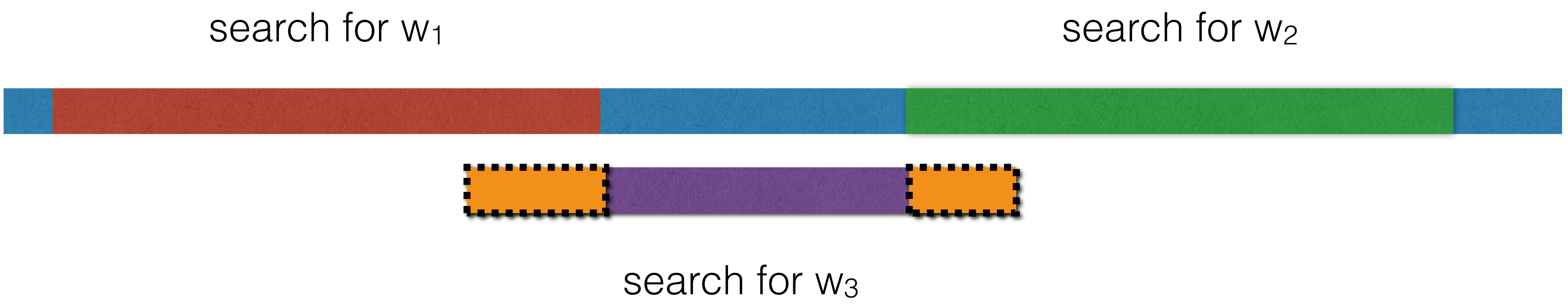


Overlap of search for w_3 = size of orange regions

- ➡ f-overlap $\Longrightarrow$ any search touches f common bits
- ➡ **intuition**: large overlaps ≈ reading more bits than necessary

read overlaps

Encrypted index in memory:



Overlap of search for w_3 = size of orange regions

- ➡ f-overlap $\Longrightarrow$ any search touches f common bits
- ➡ **intuition**: large overlaps ≈ reading more bits than necessary
- ➡ small overlap in known constructions (e.g. hash table access)

our results: lower bound (formal)

Let N = no. postings in input index

Theorem: No secure searchable encryption can have all 3:

1. $O(N)$-size encrypted index
2. $O(1)$-locality
3. $O(1)$-overlap on searches

- ➡ super-linear blow-up in storage/locality or highly overlapping reads
- ➡ in paper: smooth trade-off
- ✳ can be circumvented by changing security def [CJJJKRS]

memory utilization of constructions

N = no. postings in input index, R = no. postings in search

	Enc Ind Size	Overlap	Locality
lower bound: 1 of	$\omega(N)$	$\omega(1)$	$\omega(1)$
[CGKO,KPR,…]	N	1	R
[CK]	N^2	1	1

memory utilization of constructions

N = no. postings in input index, R = no. postings in search

	Enc Ind Size	Overlap	Locality
lower bound: 1 of	$\omega(N)$	$\omega(1)$	$\omega(1)$
[CGKO,KPR,…]	N	1	R
[CK]	N^2	1	1
trivial "read all"	N	N	1

memory utilization of constructions

N = no. postings in input index, R = no. postings in search

	Enc Ind Size	Overlap	Locality
lower bound: 1 of	$\omega(N)$	$\omega(1)$	$\omega(1)$
[CGKO,KPR,…]	N	1	R
[CK]	N^2	1	1
trivial “read all”	N	N	1
new construction	N log N	log N	log N

memory utilization of constructions

N = no. postings in input index, R = no. postings in search

	Enc Ind Size	Overlap	Locality
lower bound: 1 of	$\omega(N)$	$\omega(1)$	$\omega(1)$
[CGKO,KPR,…]	N	1	R
[CK]	N^2	1	1
trivial “read all”	N	N	1
new construction	$N \log N$	$\log N$	$\log N$

➡ open problem: get closer to lower bound

Rest of talk

- a prior construction and why it cannot be “localized”

- lower bound approach

[CGKO] construction

Encrypted Index Generation Step 1:

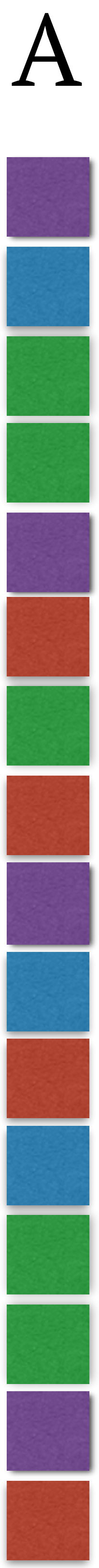
- derive per-term encryption keys: $K_i = \mathrm{PRF}(w_i)$

- encrypt individual postings under respective keys

term	postings
Rutgers	4, 9,37
Admissions	9,37,93,94,95
Committee	8,37,89,90
Accept	4,37,62,75

term	postings
K	🟦, 🟦, 🟦
K	🟩, 🟩, 🟩, 🟩, 🟩
K	🟥, 🟥, 🟥, 🟥
K	🟪, 🟪, 🟪, 🟪

[CGKO] construction



Encrypted Index Generation Step 2:

1. put ciphertexts in random order in array A
2. link together postings lists with encrypted pointers (encrypted under K_i)
3. encrypted index = A

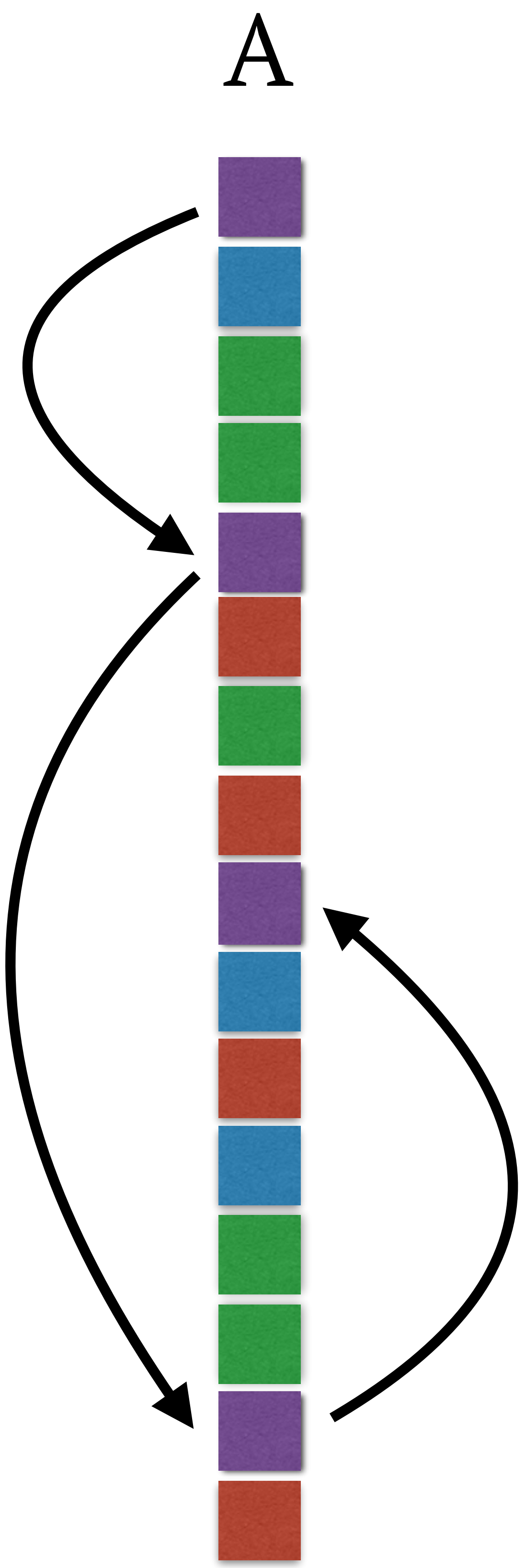
[CGKO] construction

Encrypted Index Generation Step 2:

1. put ciphertexts in random order in array A

2. link together postings lists with encrypted pointers (encrypted under $\mathrm{K_i}$)

3. encrypted index = A



(example with pointers for word “Accept”)

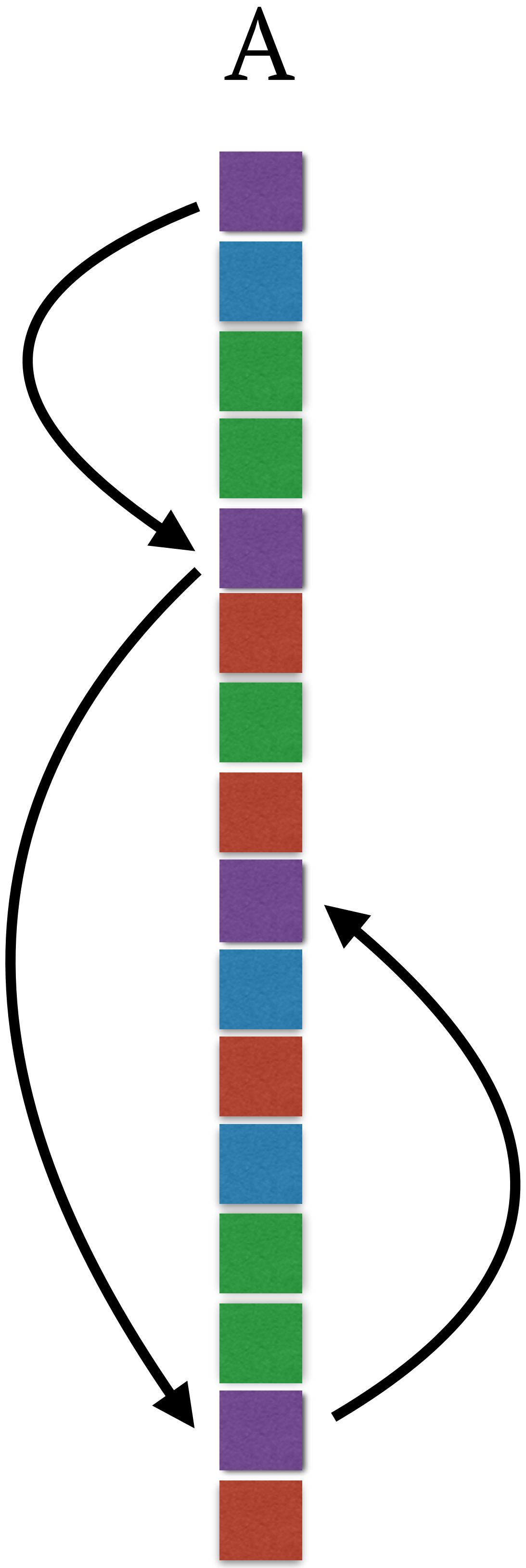
[CGKO] construction: searching

token generation for w:

- re-derive key K = PRF(w)
- token = K

server search using token:

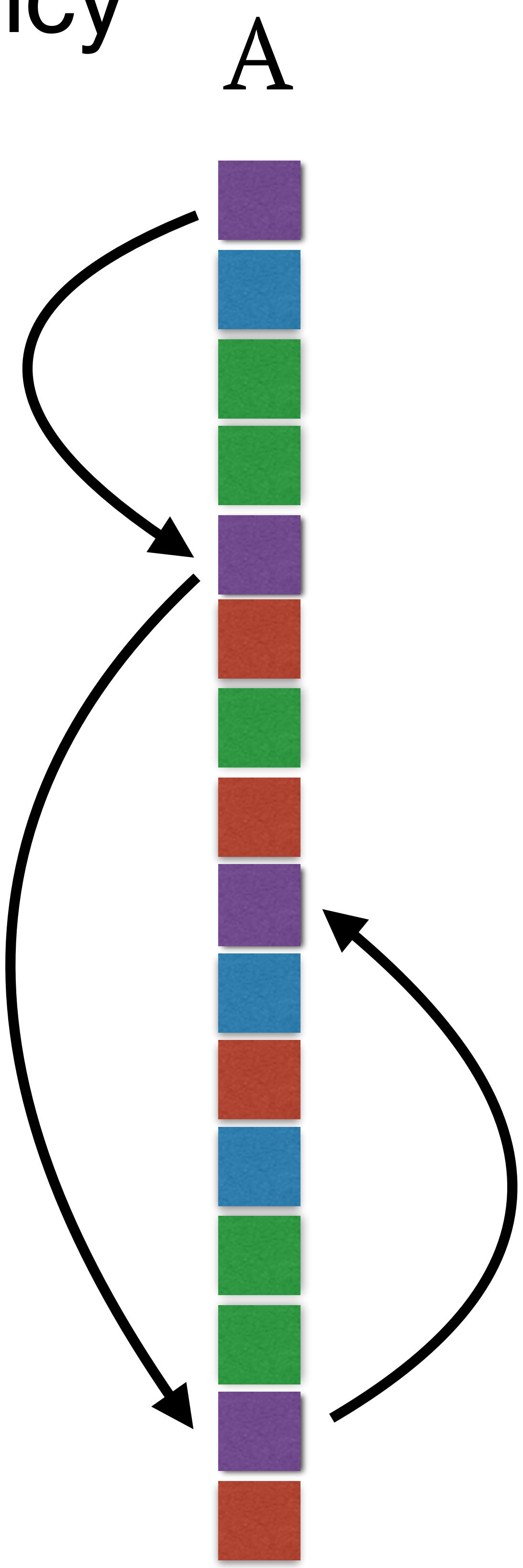
- step through list, decrypt postings/ pointers with K



[CGKO] construction: memory efficiency

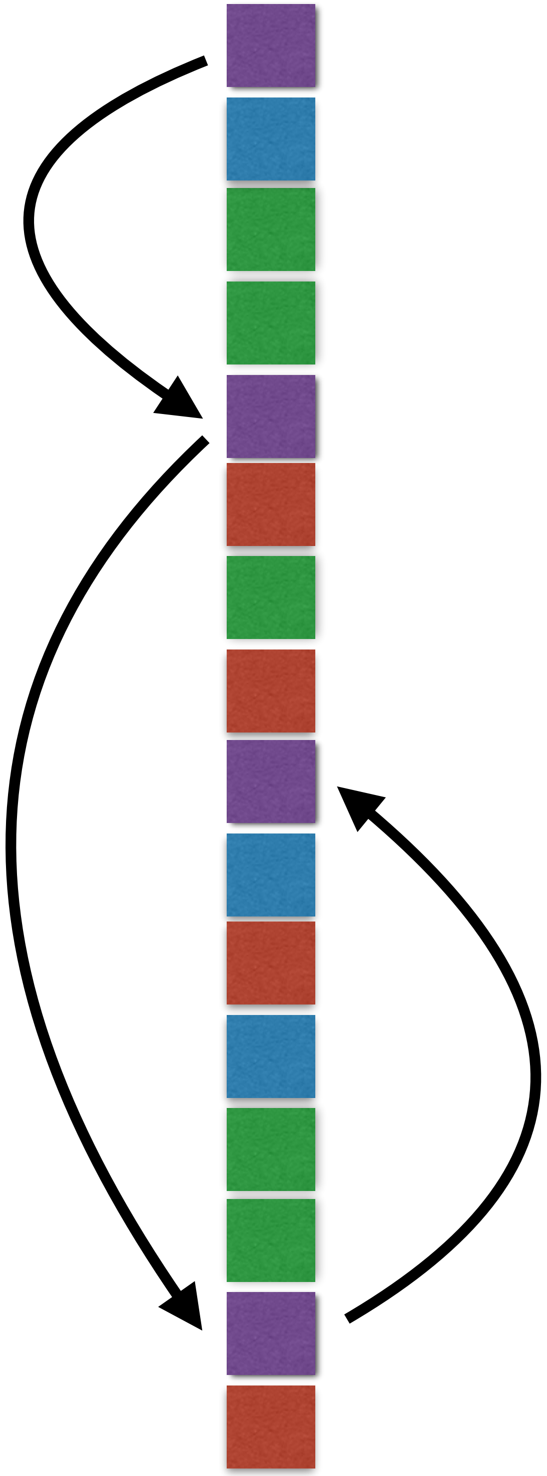
Memory utilization:

- O(N) size index
- O(R) locality for search w/ R postings
- O(1) read overlaps



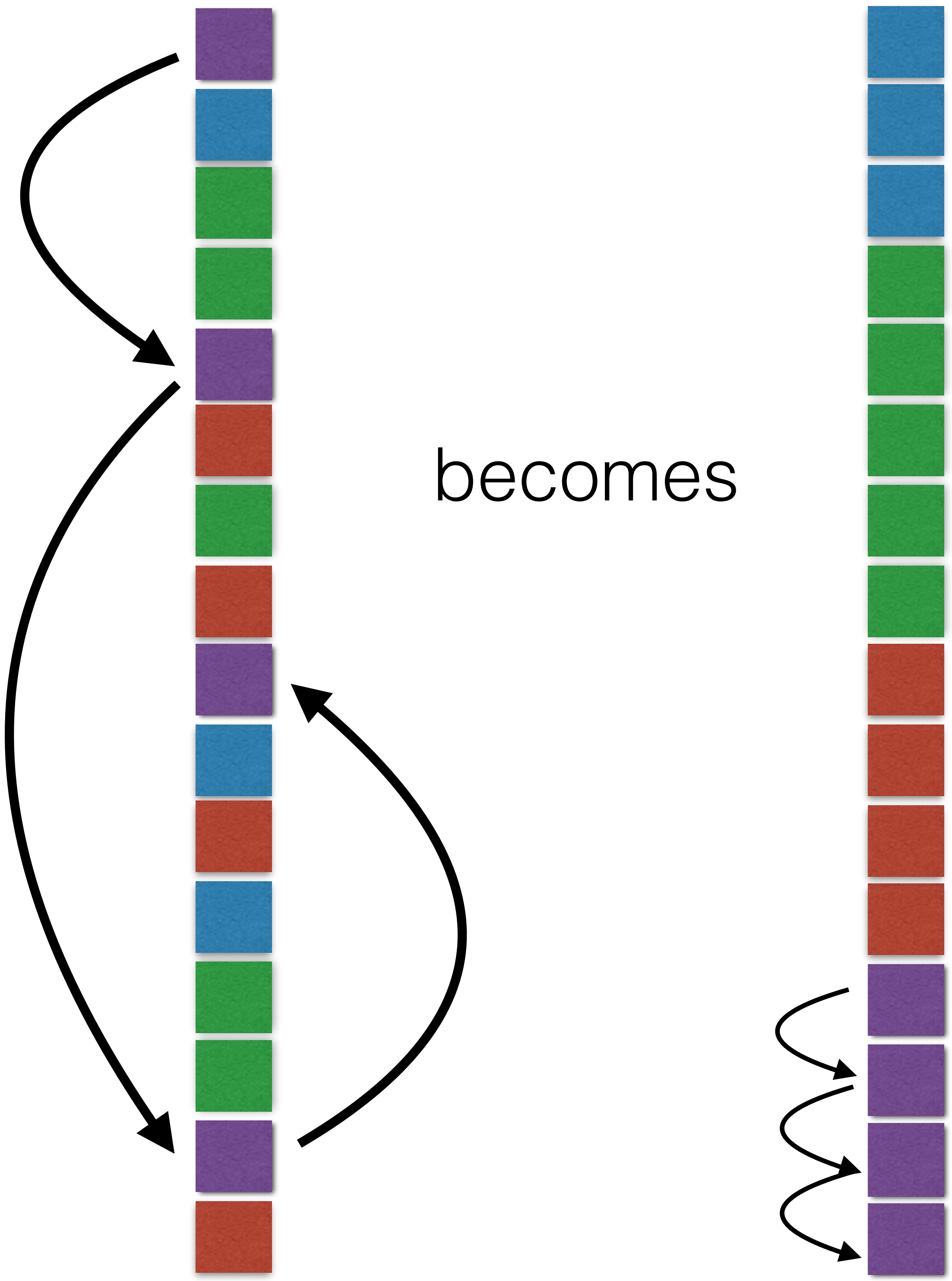
suppose we try to make construction "local"

- ➡ store encrypted postings lists together.



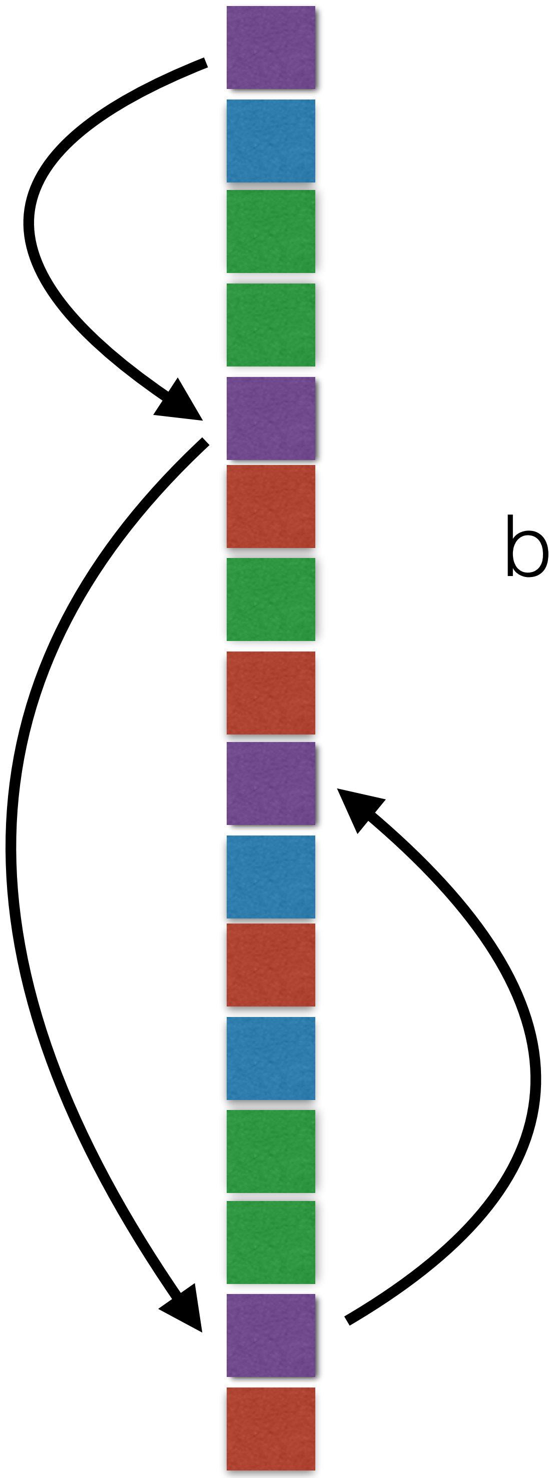
suppose we try to make construction “local”

➡ store encrypted postings lists together.

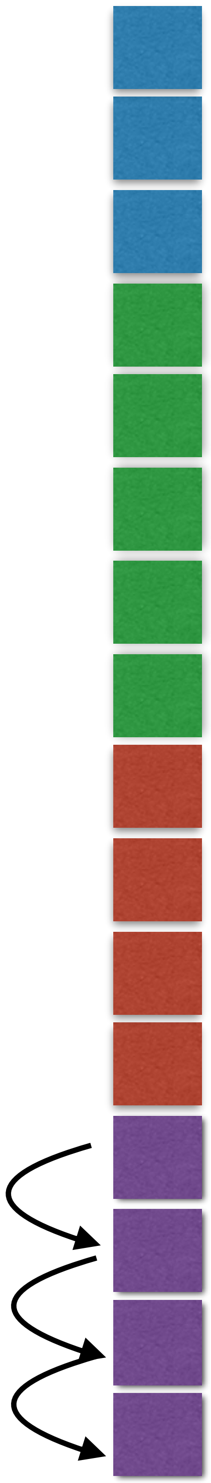


suppose we try to make construction "local"

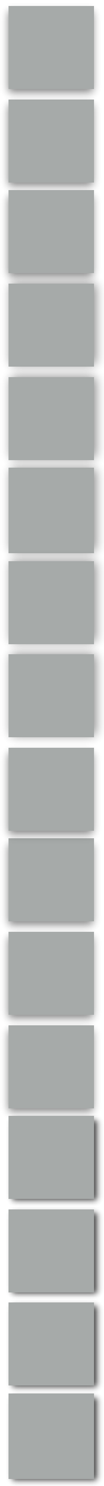
➡ store encrypted postings lists together.



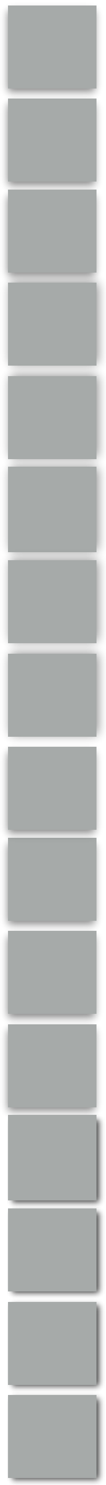
becomes



which looks like

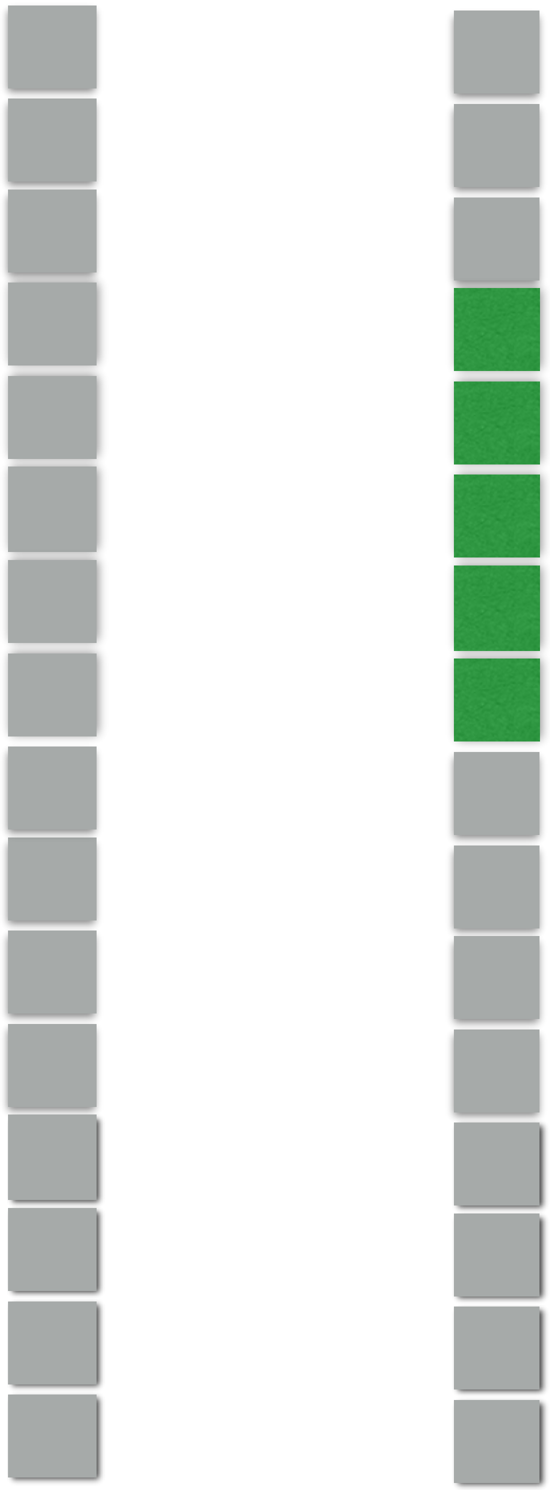


server can observe memory touched during searches:

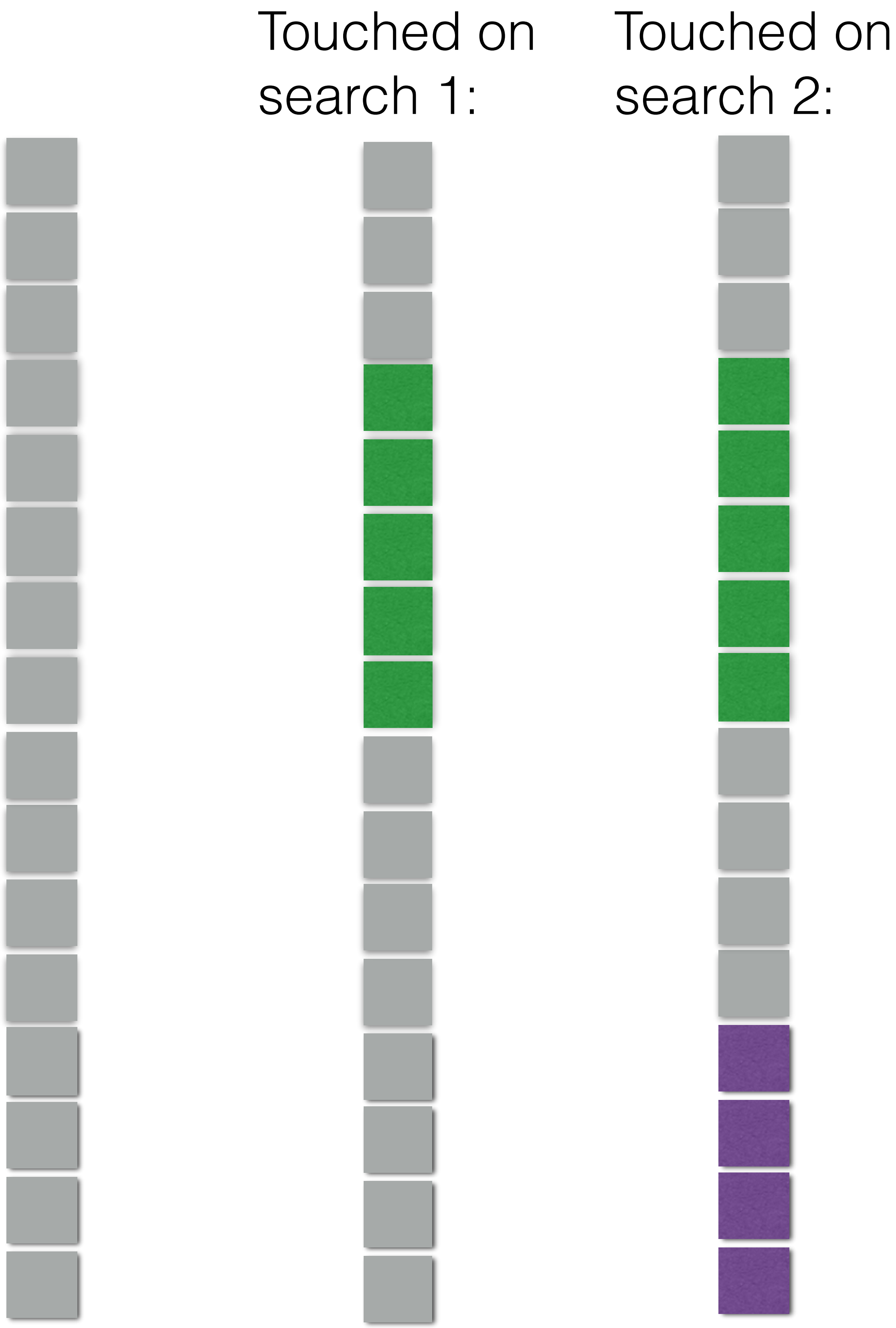


server can observe memory touched during searches:

Touched on search 1:



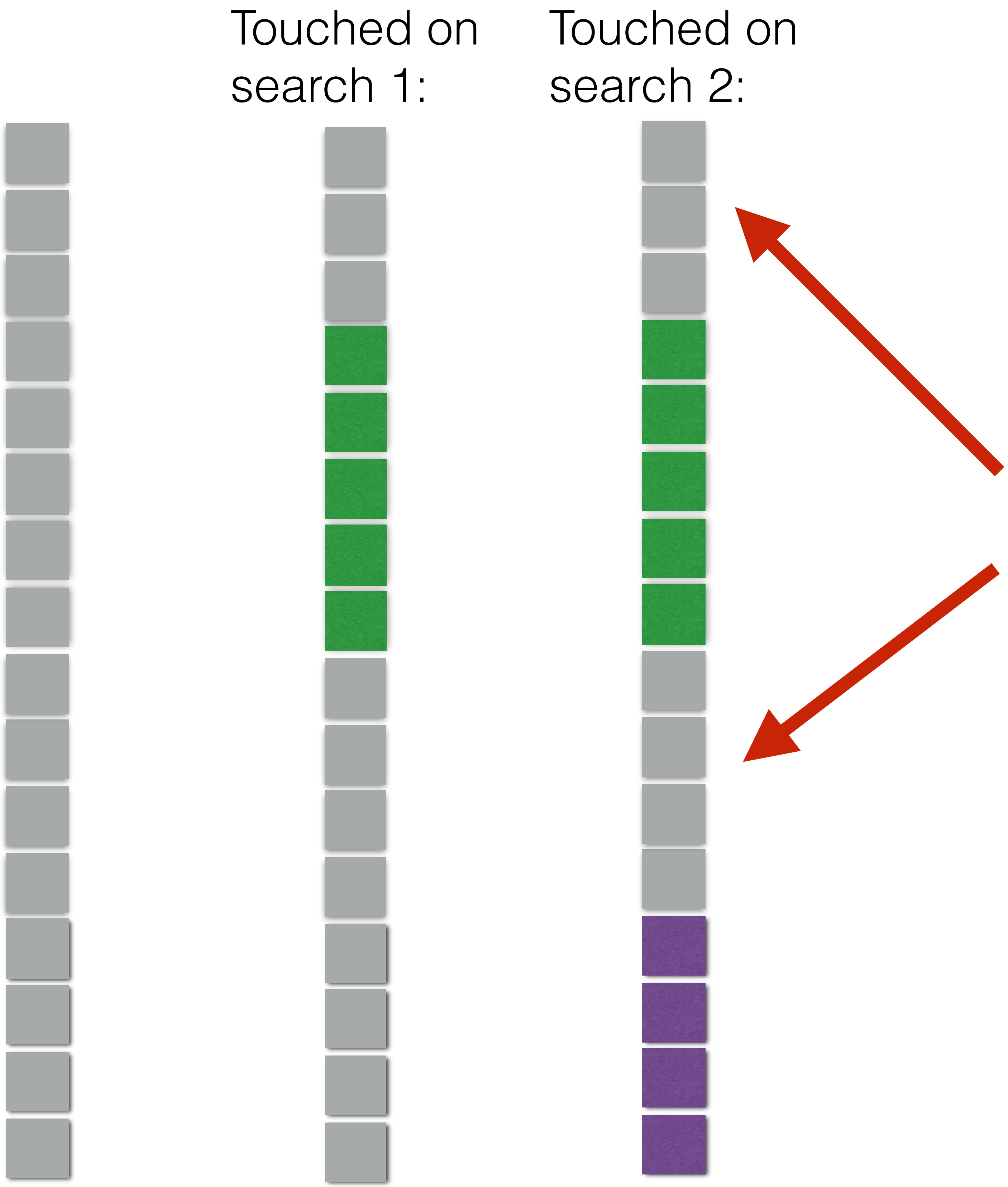
server can observe memory touched during searches:



Touched on search 1:

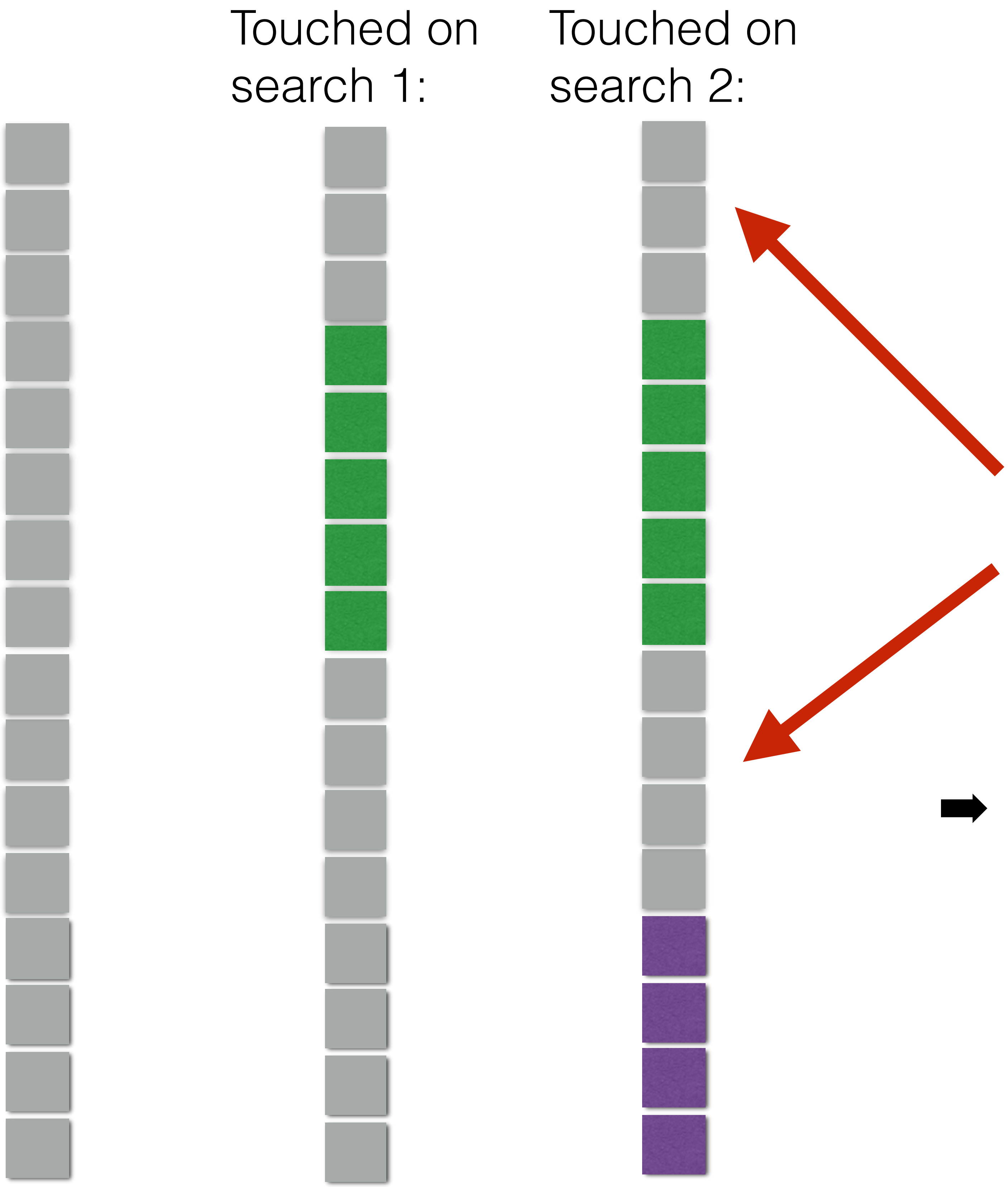
Touched on search 2:

server can observe memory touched during searches:



composition of untouched regions reveals info about unopened part of index!

server can observe memory touched during searches:



composition of untouched regions reveals info about unopened part of index!

➡ e.g. 7 remaining spots do not correspond to a single postings list

proof intuition

- adapt this attack to work against any scheme

proof intuition

- adapt this attack to work against any scheme
  - distinguish two crafted indexes by observing reads and testing for biases in their distribution

proof intuition

- Lemma 1: If scheme is secure, then memory touched during a O(1)-local search satisfies a mild pseudorandomness condition

proof intuition

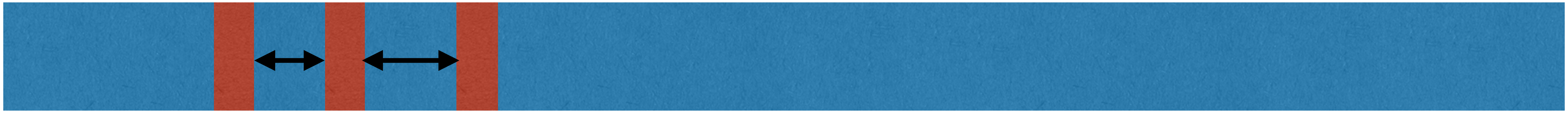
- Lemma 1: If scheme is secure, then memory touched during a O(1)-local search satisfies a mild pseudorandomness condition

proof intuition

- Lemma 1: If scheme is secure, then memory touched during a O(1)-local search satisfies a mild pseudorandomness condition
- Lemma 2: Pseudorandom reads will have small gaps often between contiguous regions often.

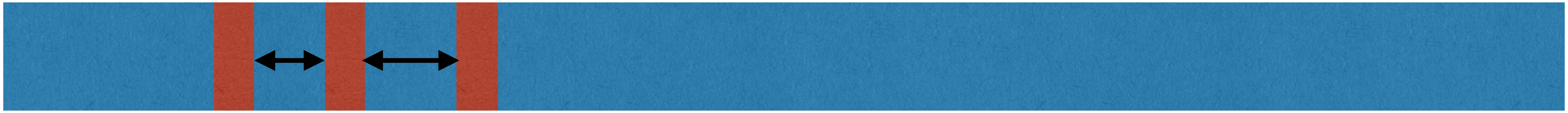
proof intuition

- Lemma 1: If scheme is secure, then memory touched during a O(1)-local search satisfies a mild pseudorandomness condition
- Lemma 2: Pseudorandom reads will have small gaps often between contiguous regions often.



proof intuition

- Lemma 1: If scheme is secure, then memory touched during a O(1)-local search satisfies a mild pseudorandomness condition
- Lemma 2: Pseudorandom reads will have small gaps often between contiguous regions often.



- small gaps can't hold contiguous intervals for other searches, so gap space is "dead" for searches with larger postings lists
- delicate argument to formalize, requires further techniques for full theorem

summary

- ➡ first results relating i/o efficiency and security of crypto primitive

summary

➡ first results relating i/o efficiency and security of crypto primitive

➡ unconditional lower bounds via new proof technique

- completely different from known i/o lower bounds

summary

➡ first results relating i/o efficiency and security of crypto primitive

➡ unconditional lower bounds via new proof technique

- completely different from known i/o lower bounds

➡ improved theoretical i/o efficiency of prior work

summary

- ➡ first results relating i/o efficiency and security of crypto primitive
- ➡ unconditional lower bounds via new proof technique
  - completely different from known i/o lower bounds
- ➡ improved theoretical i/o efficiency of prior work

Q1: Tighten gap between upper/lower bound?

summary

- ➡ first results relating i/o efficiency and security of crypto primitive
- ➡ unconditional lower bounds via new proof technique
  - completely different from known i/o lower bounds
- ➡ improved theoretical i/o efficiency of prior work

Q1: Tighten gap between upper/lower bound?

Q2: Fine-grained lower bounds?

summary

➡ first results relating i/o efficiency and security of crypto primitive

➡ unconditional lower bounds via new proof technique

- completely different from known i/o lower bounds

➡ improved theoretical i/o efficiency of prior work

Q1: Tighten gap between upper/lower bound?

Q2: Fine-grained lower bounds?

Q3: Other primitives where i/o efficiency dominates?

Thanks!