Searchable Encryption

New Constructions of Encrypted Databases

Slides at https://r.bost.fyi/phd

Raphael Bost - 8/01/2017

Searchable Encryption

Outsource data

- Securely
- Keep search functionalities
- Aimed at efficiency
- … we have to leak some information …
- … and this can lead to devastating attacks

An example: property preserving encryption

Deterministic encryption, Order Preserving Encryption

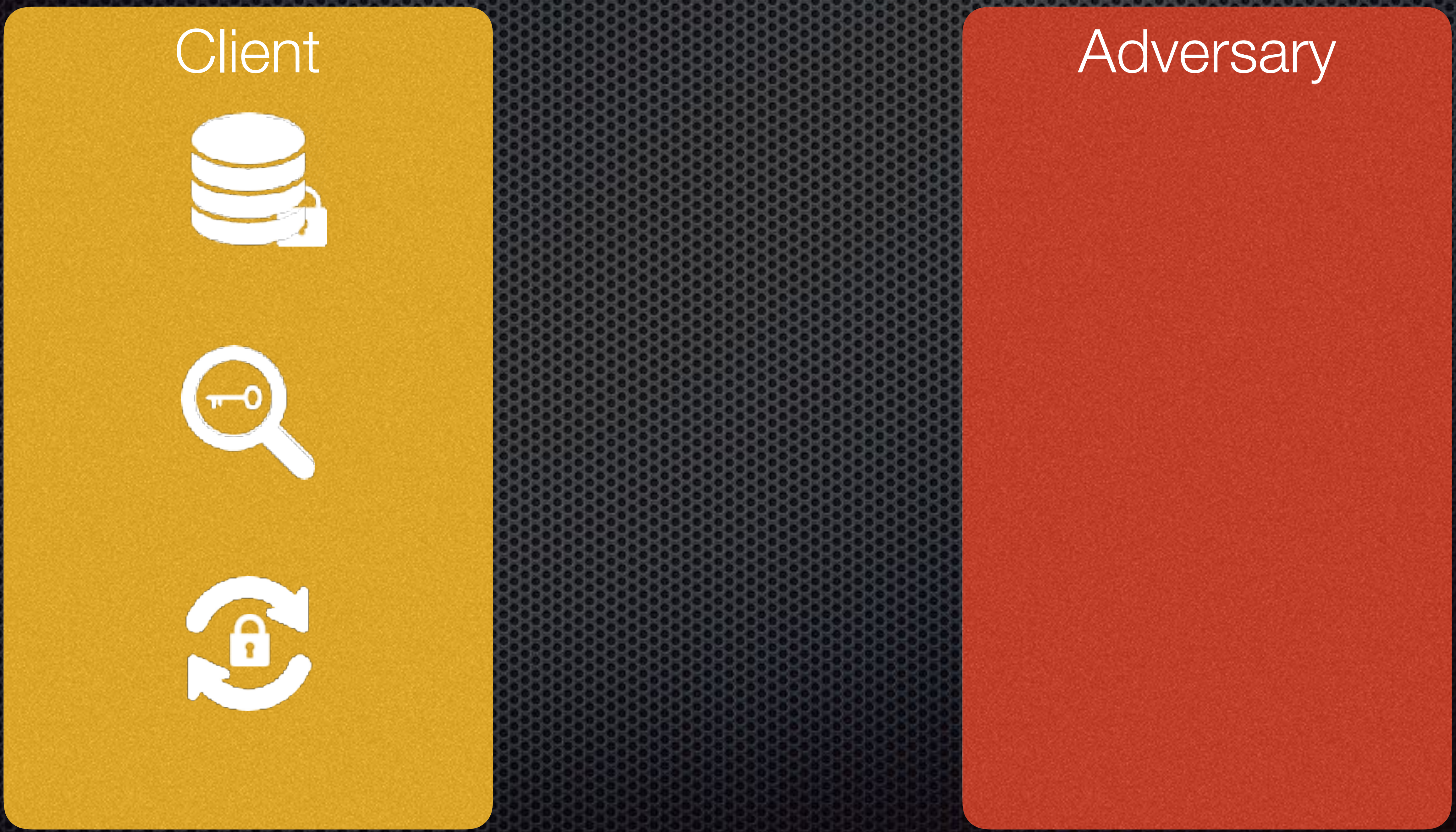
✓ Legacy compatible (works on top of unencrypted DB)

✓ Very efficient

✗ Not secure in practice (frequency analysis)

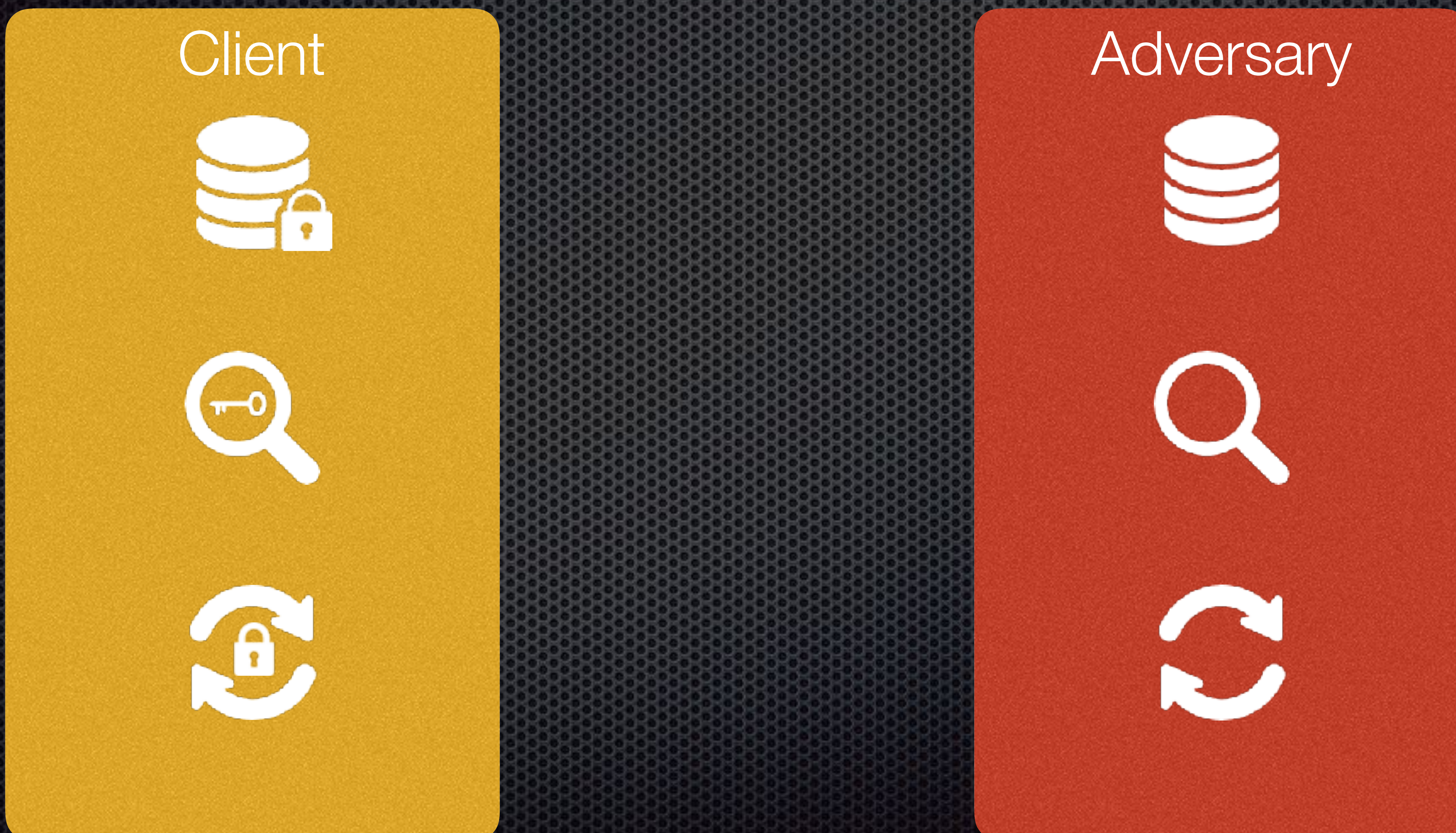
Security of SE

- Everything the server learns can be computed from the leakage



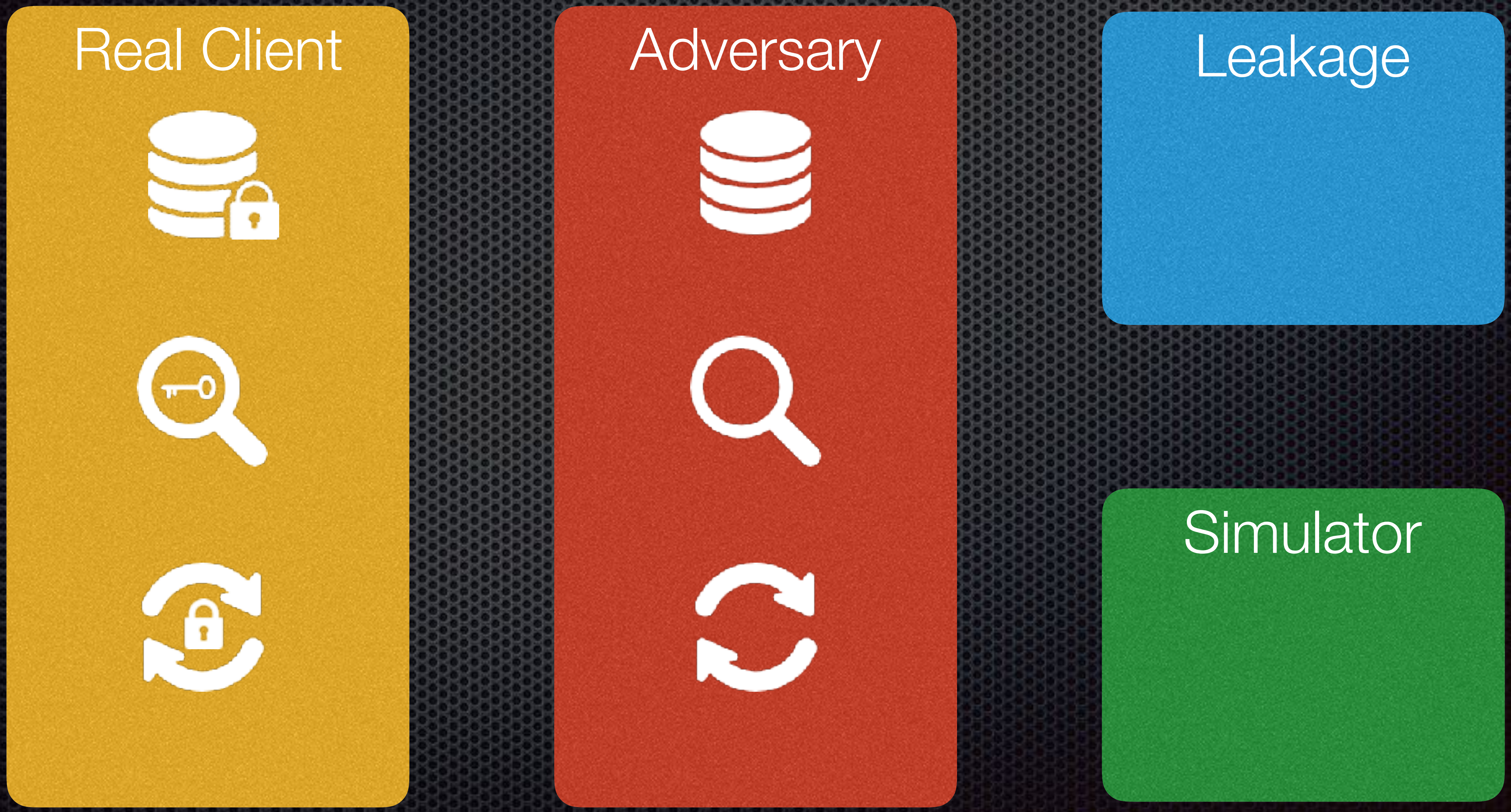
Security of SE

- Everything the server learns can be computed from the leakage



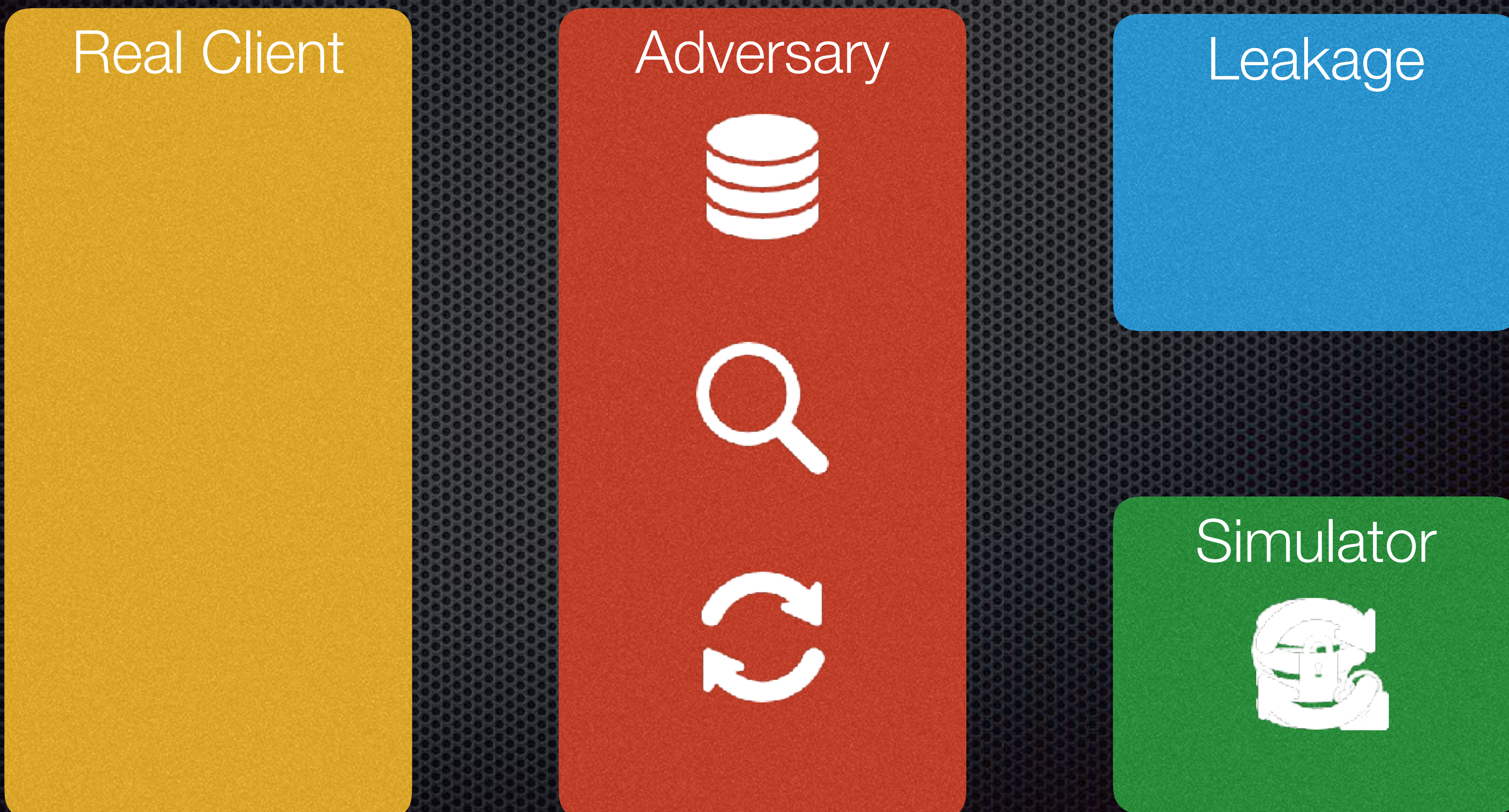
Security of SE

- Everything the server learns can be computed from the leakage



Security of SE

- Everything the server learns can be computed from the leakage



Security of SE

- Everything the server learns can be computed from the leakage

Real Client

Adversary

Leakage

Ideal World

Simulator

Examples of leakage

- After a search, the user will access the matching documents. This will reveal the search result.
- When the user searches for the same keyword twice, the server might learn that the query has been repeated.
- In both cases, trying to get rid of this leakage is expensive

An explicit tradeoff between security and performance

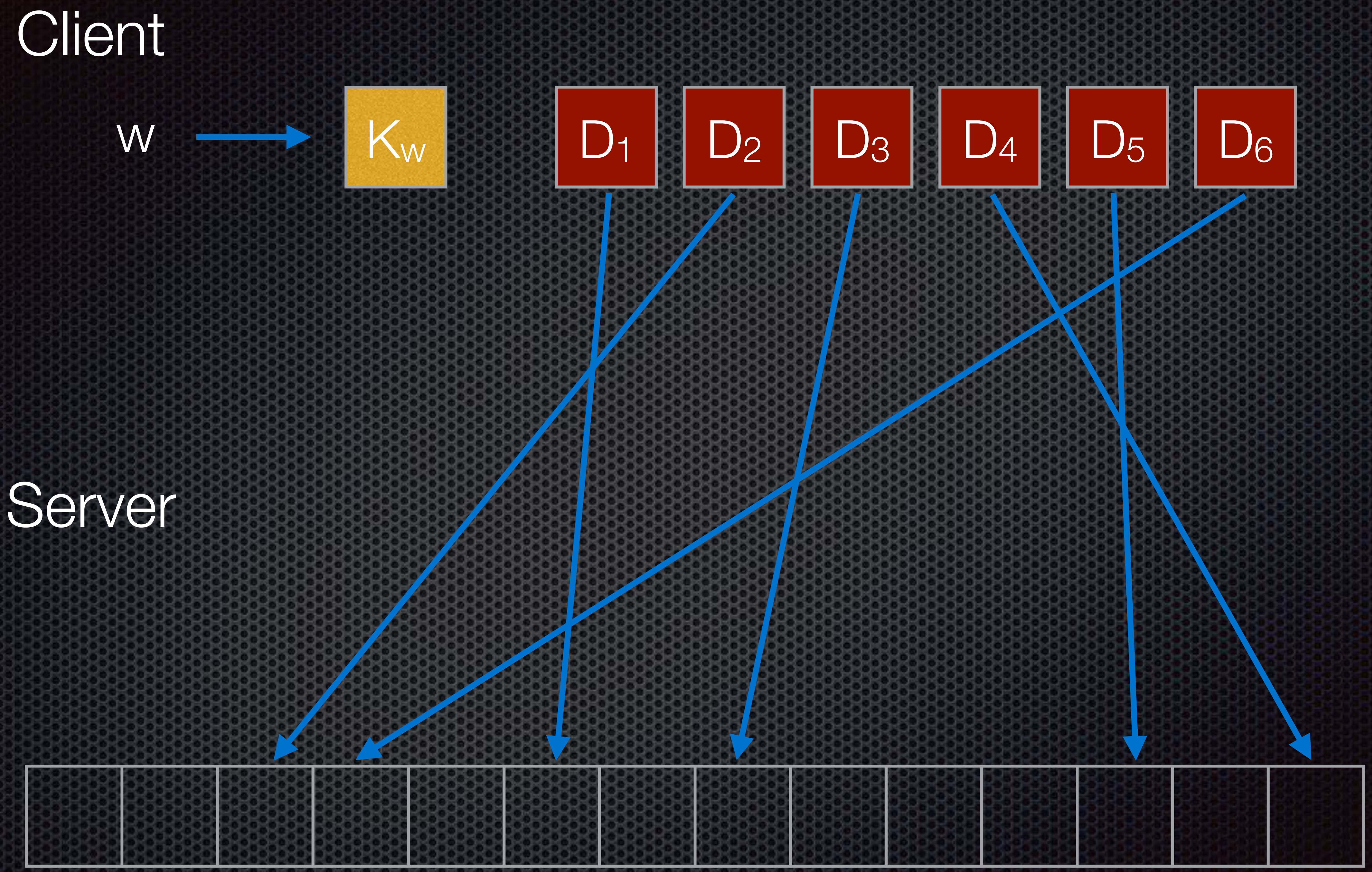
- Oblivious RAM lower bound: if one wants to hide the access pattern to a memory of size N, the computational overhead is

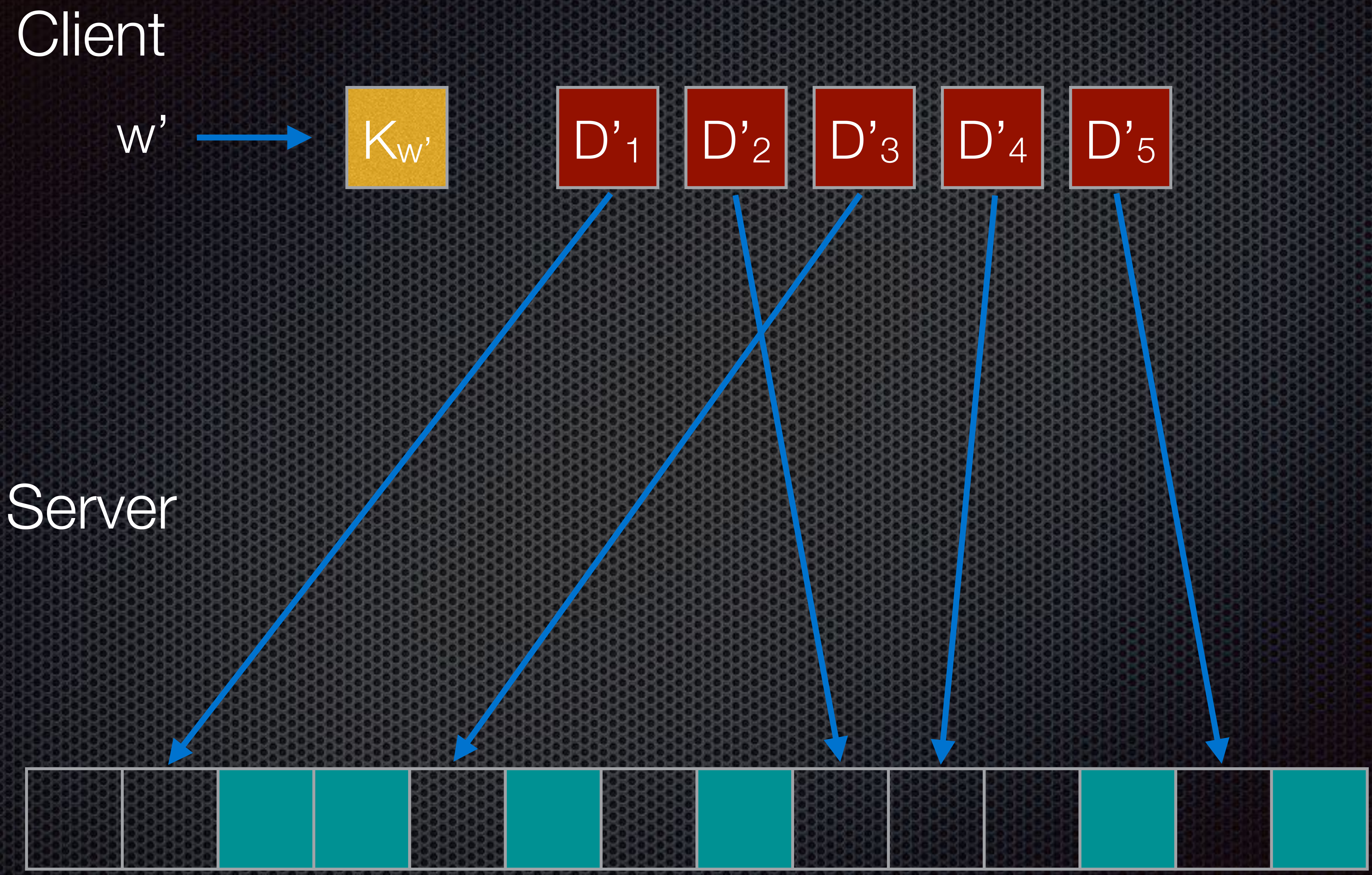
$$\Omega\left(\frac{\log N}{\log \sigma}\right)$$

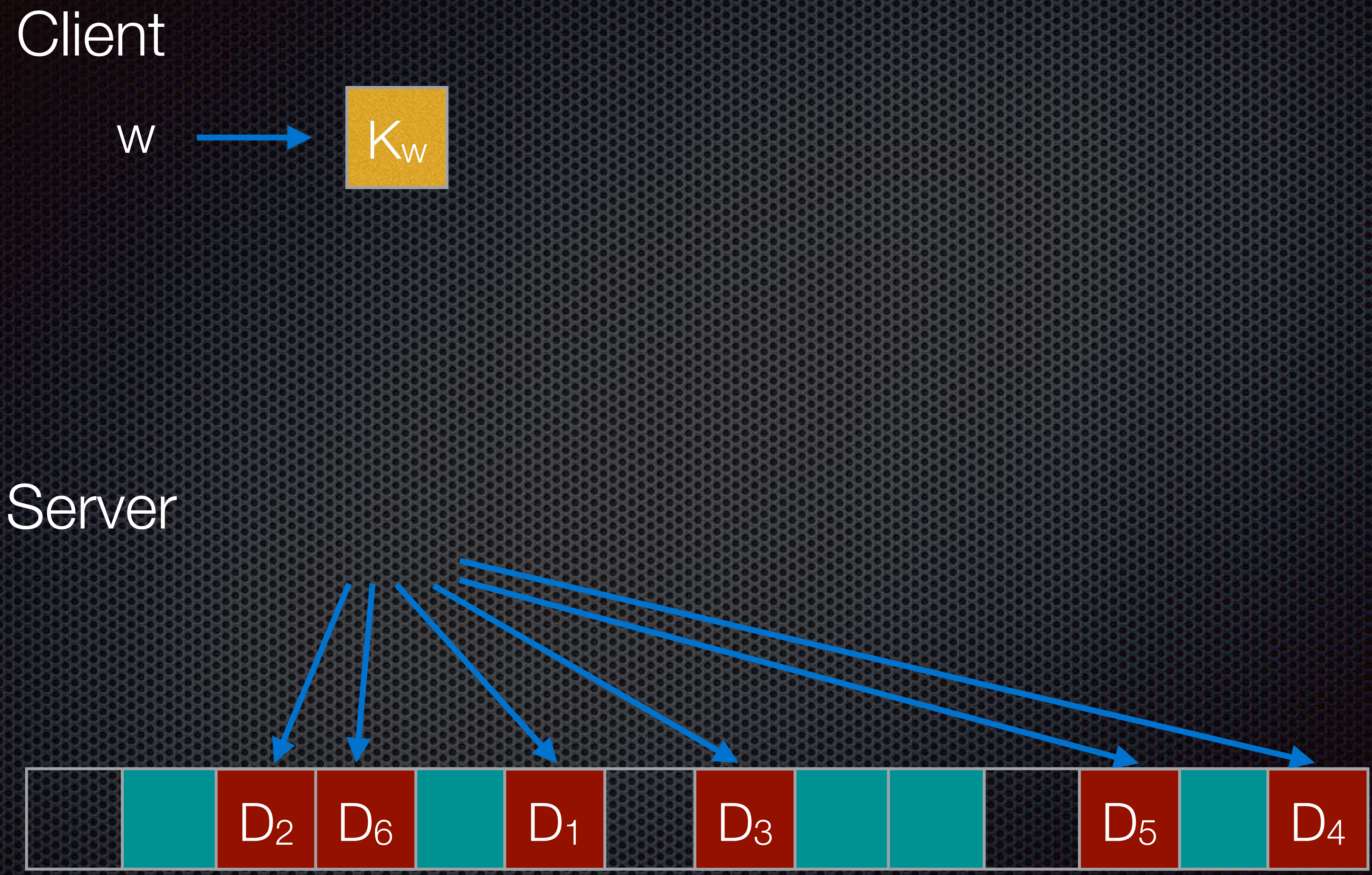
- A similar lower bound exists for searchable encryption: a search pattern-hiding SE incurs a search overhead of

$$\Omega\left(\frac{\log \binom{|DB|}{n_w}}{\log \sigma}\right)$$

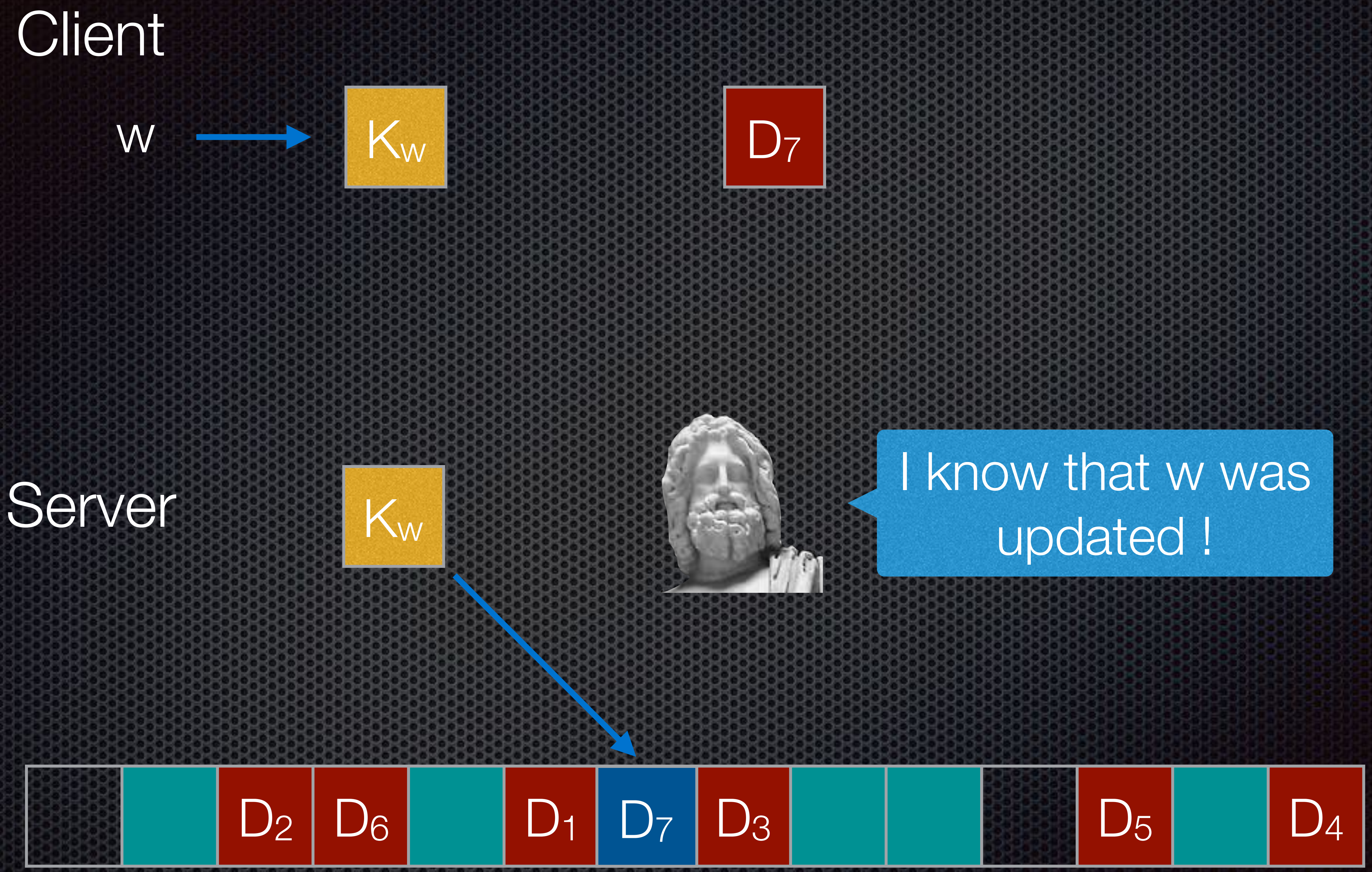
Constructing encrypted databases


Client
w
K_w
D_1
D_2
D_3
D_4
D_5
D_6
Server


Client
w'
K_w'
D'1
D'2
D'3
D'4
D'5
Server



Client
w
Kw
Server
D2
D6
D1
D3
D5
D4


Client
w
K_w
D_7
Server
K_w
I know that w was updated !
D_2
D_6
D_1
D_7
D_3
D_5
D_4

File injection attacks [ZKP'16]

- Insert purposely crafted documents in the DB (*e.g.* spam for encrypted emails)

log |W| injected documents

Active adaptive attacks

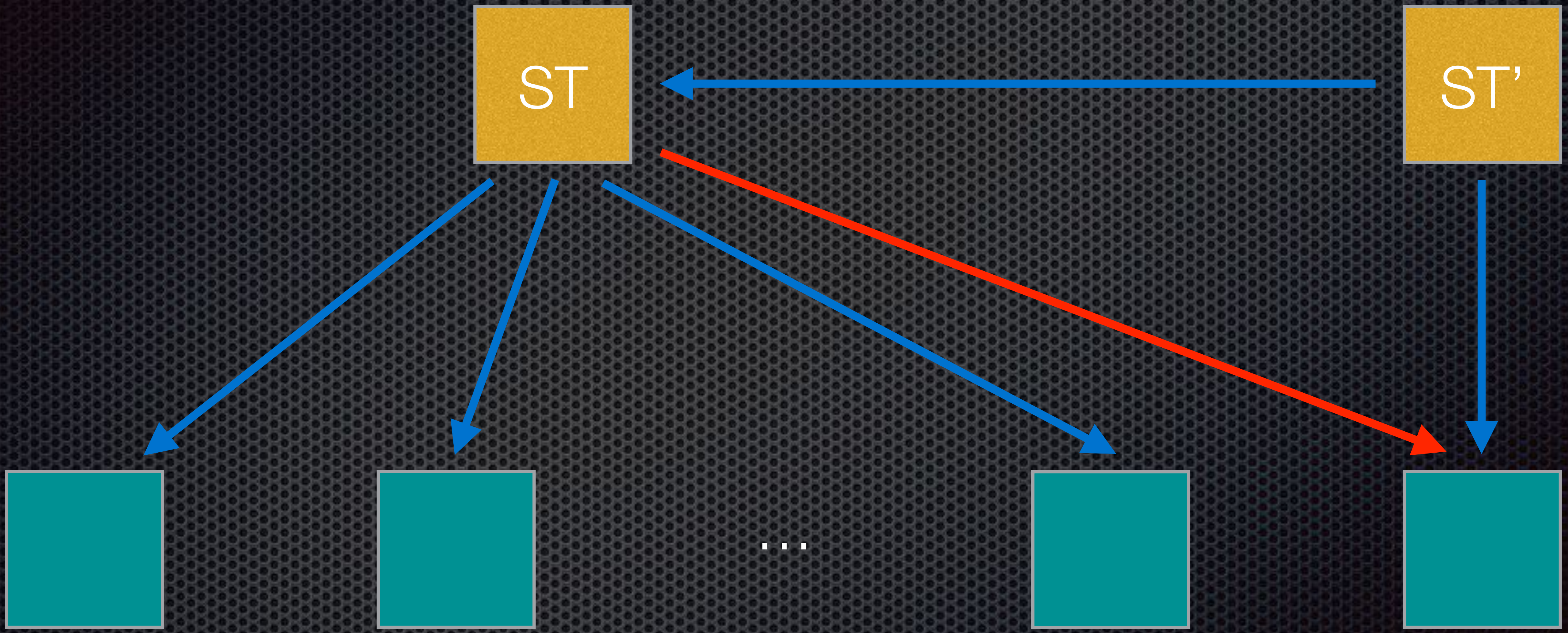
- These adaptive attacks use the update leakage
- We need SE schemes with oblivious updates

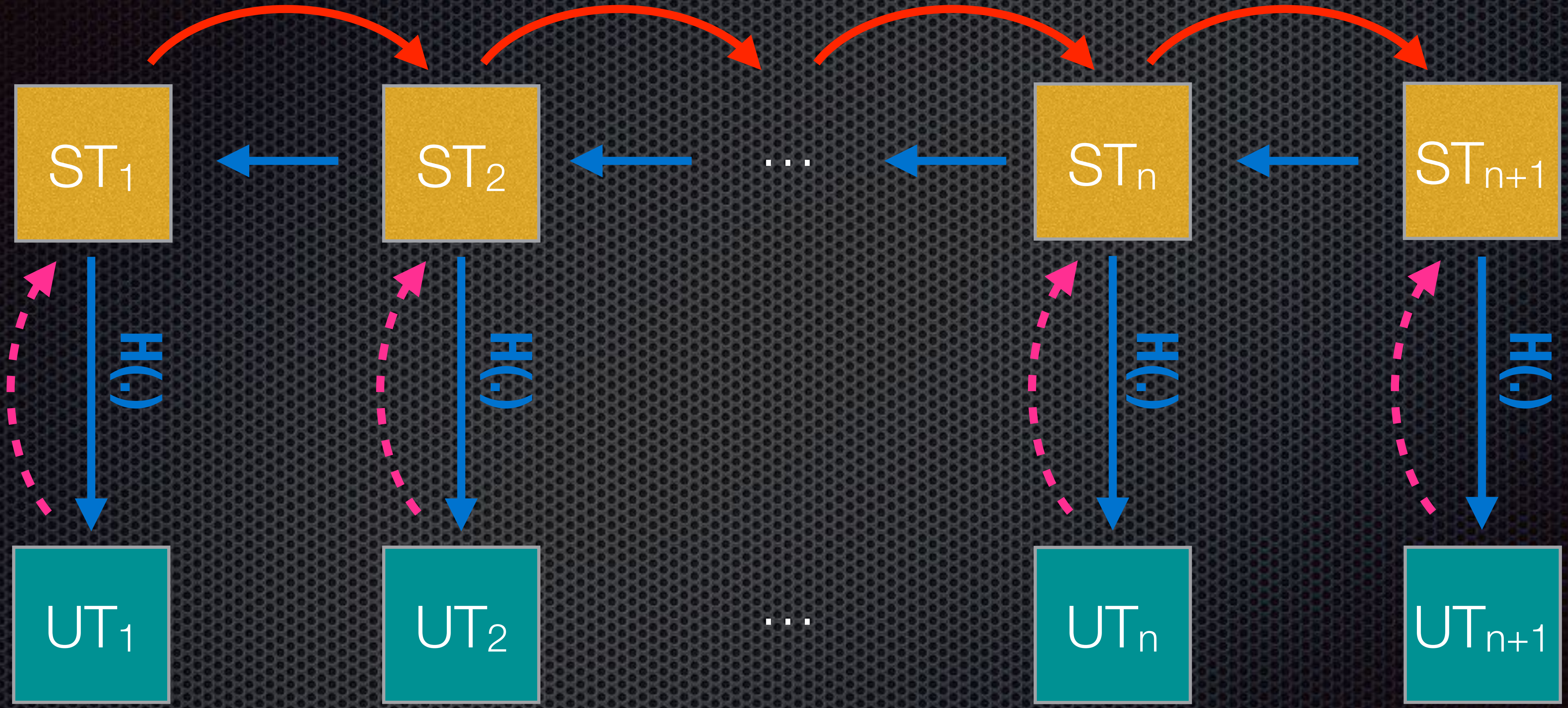
Forward Privacy

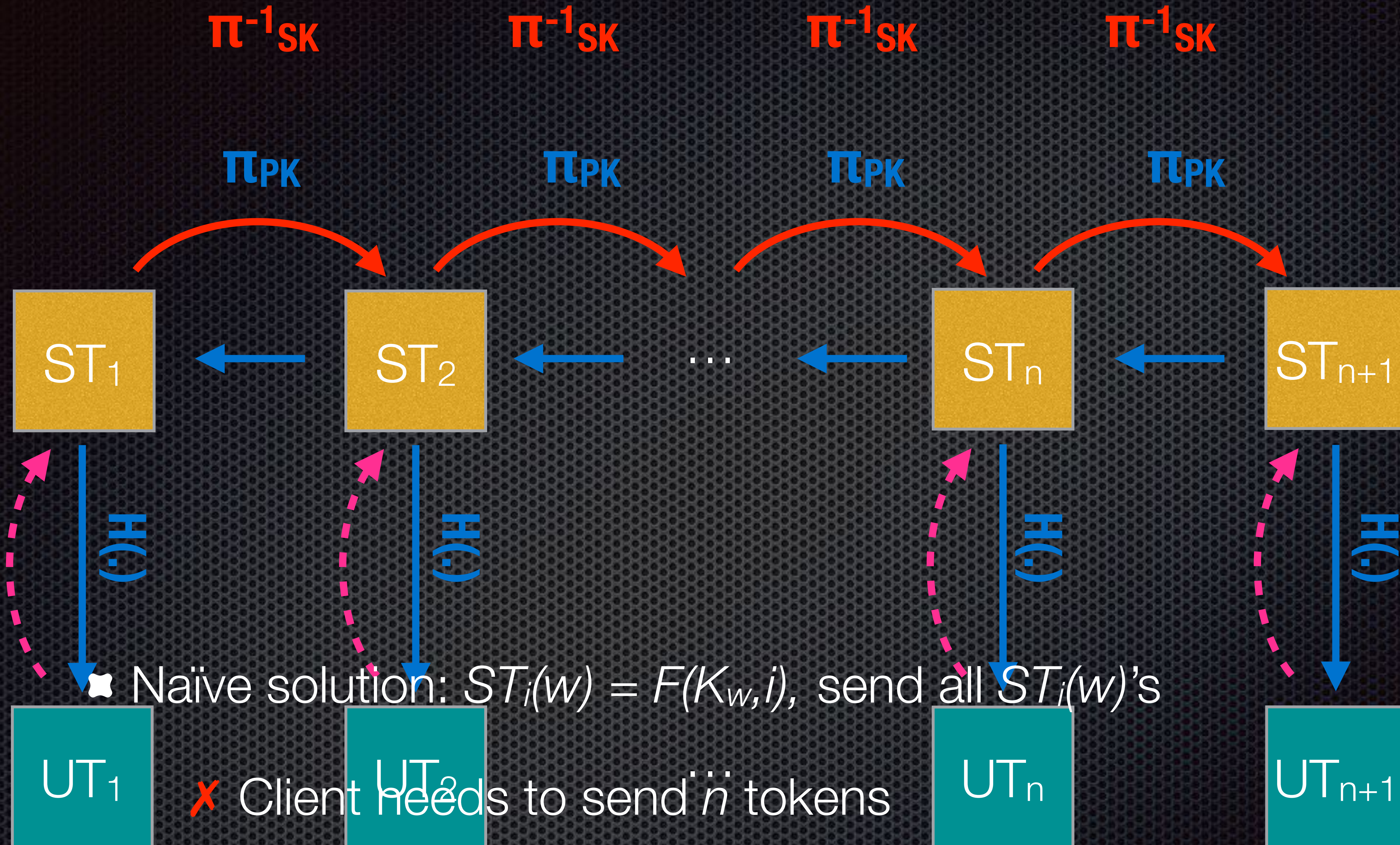
Forward privacy

- Forward private: an update does not leak any information
- Secure online build of the EDB
- Only one scheme existed so far [SPS'14]
  - ➡ ORAM-like construction
  - ✗ Inefficient updates
  - ✗ Large client storage

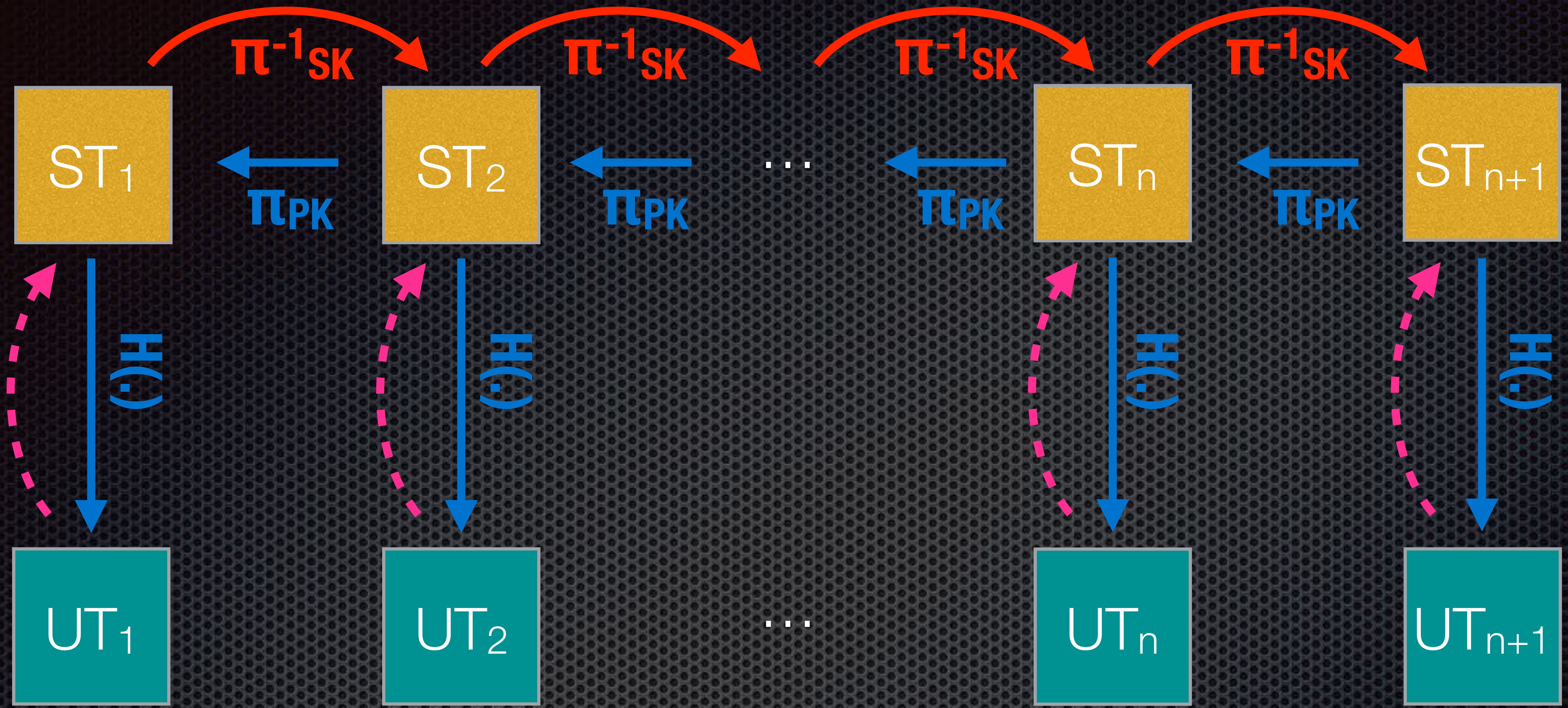
How to achieve forward privacy efficiently?


ST
ST'
....


ST1
ST2
...
STn
STn+1
H(·)
H(·)
H(·)
H(·)
UT1
UT2
...
UTn
UTn+1



- Naïve solution: $ST_i(w) = F(K_w,i)$, send all $ST_i(w)$'s
  - ✗ Client needs to send *n* tokens
- Use a trapdoor permutation (client has the secret key, server has the public key, and cannot compute the inverse)



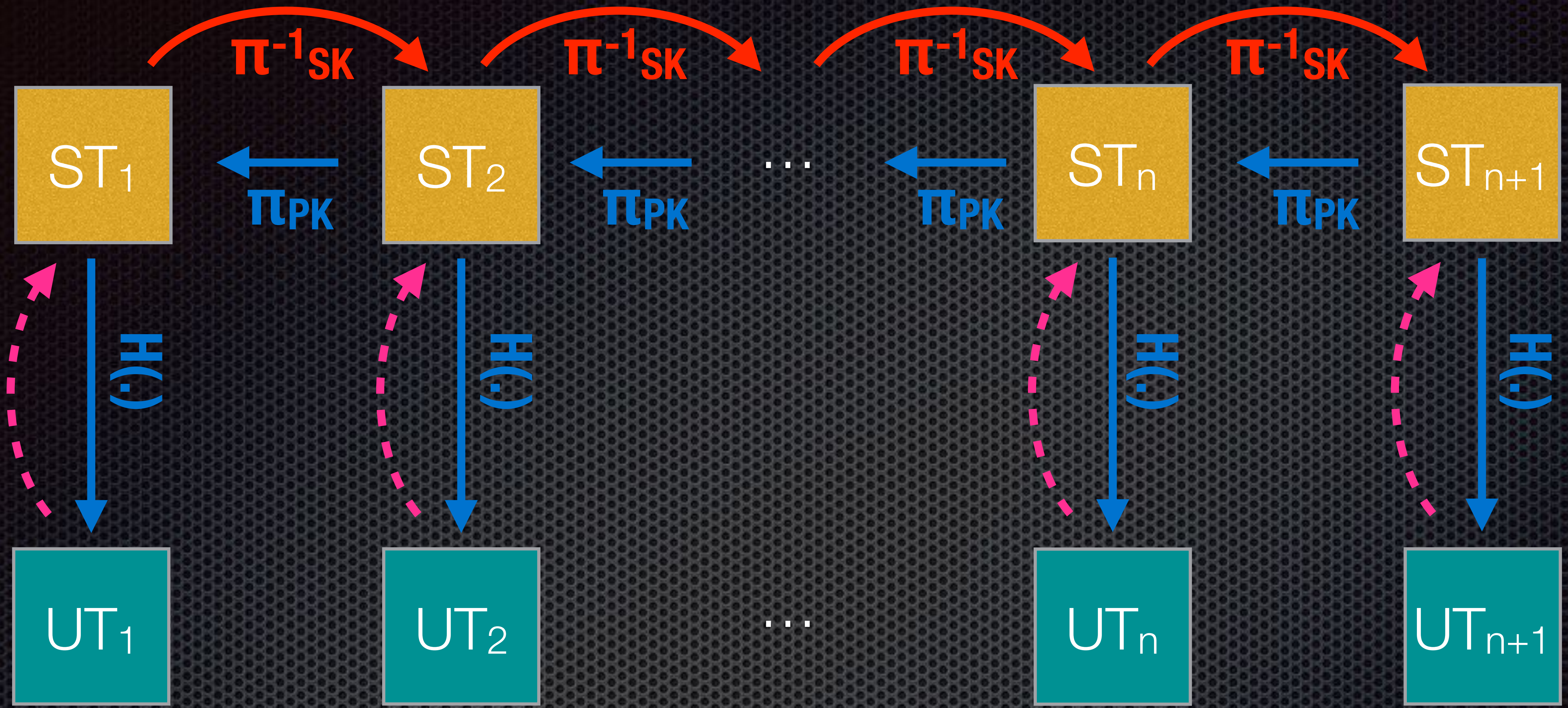
Search:

- Client: constant
- Server: # results

Update:

- Client: constant
- Server: constant

Optimal



Storage:

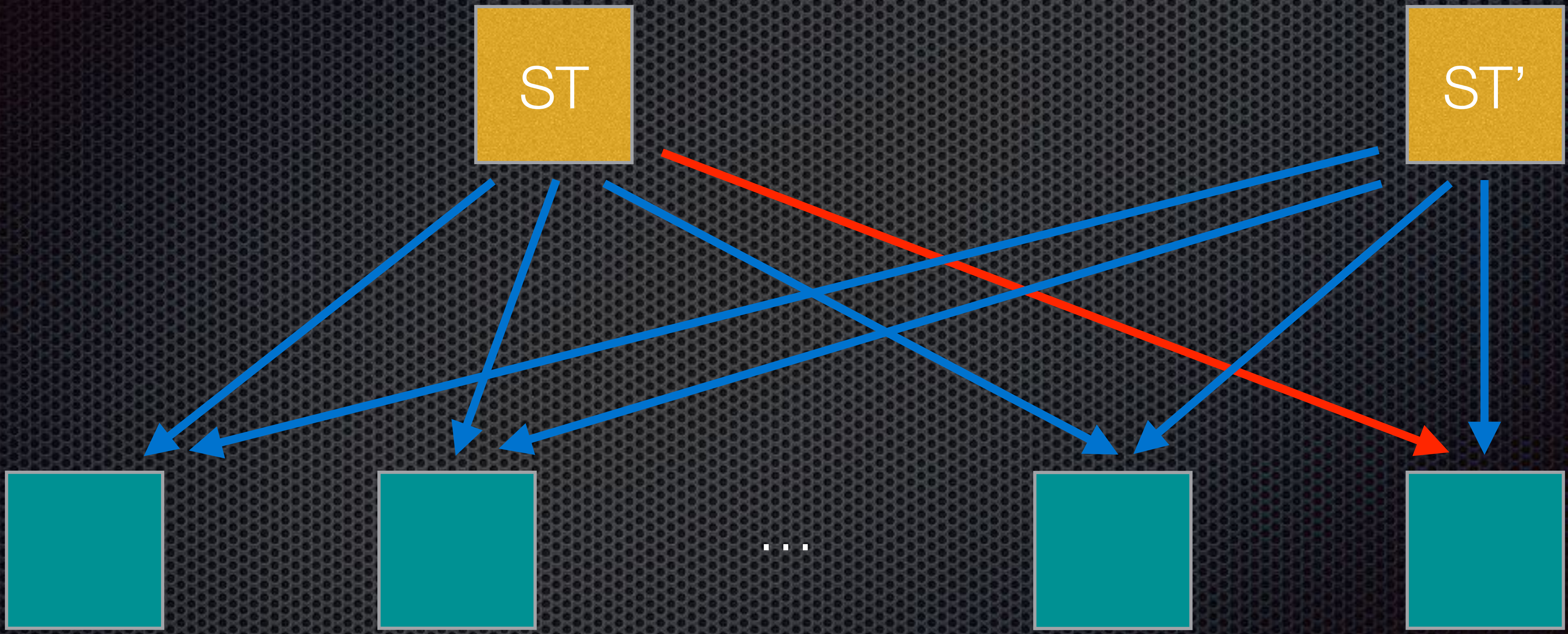
- Client: # distinct keywords
- Server: # database entries

Σοφος

- Forward private index-based scheme
- Very simple
- Efficient search (IO bounded)
- Asymptotically efficient update
  In practice, very low update throughput
  4300 updates/s — 20x slower than other work

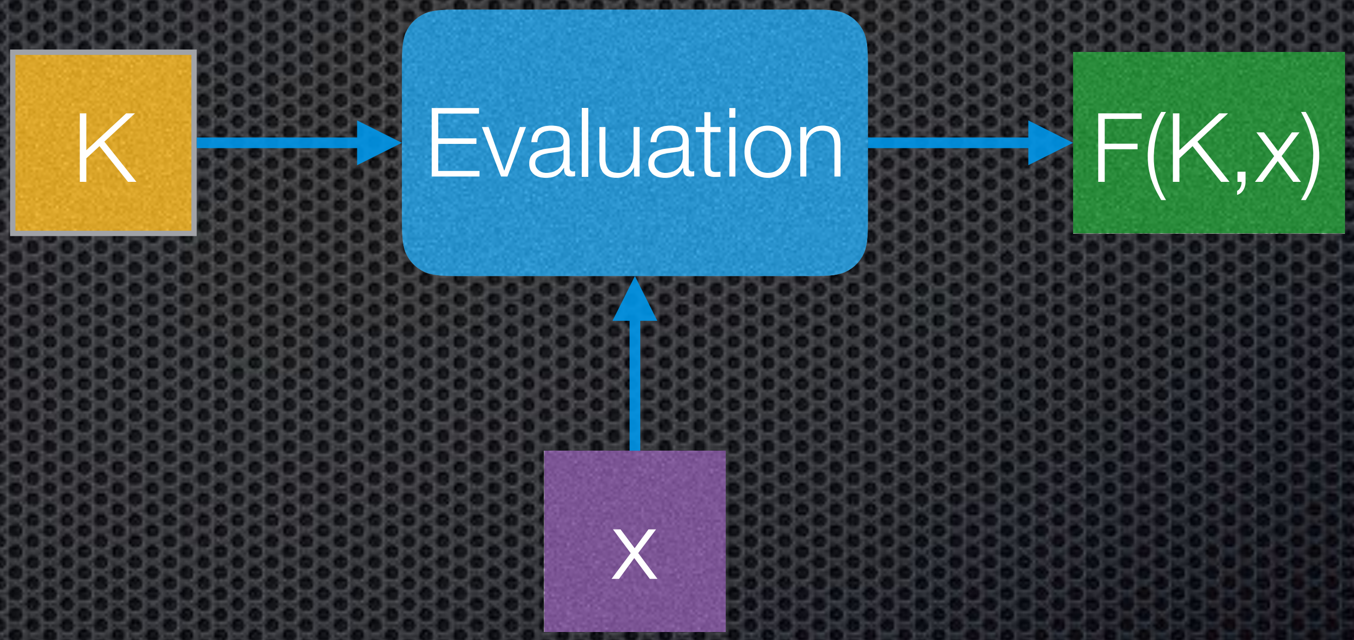
Another path towards forward privacy


ST
ST’
....


ST
ST'
. . . .

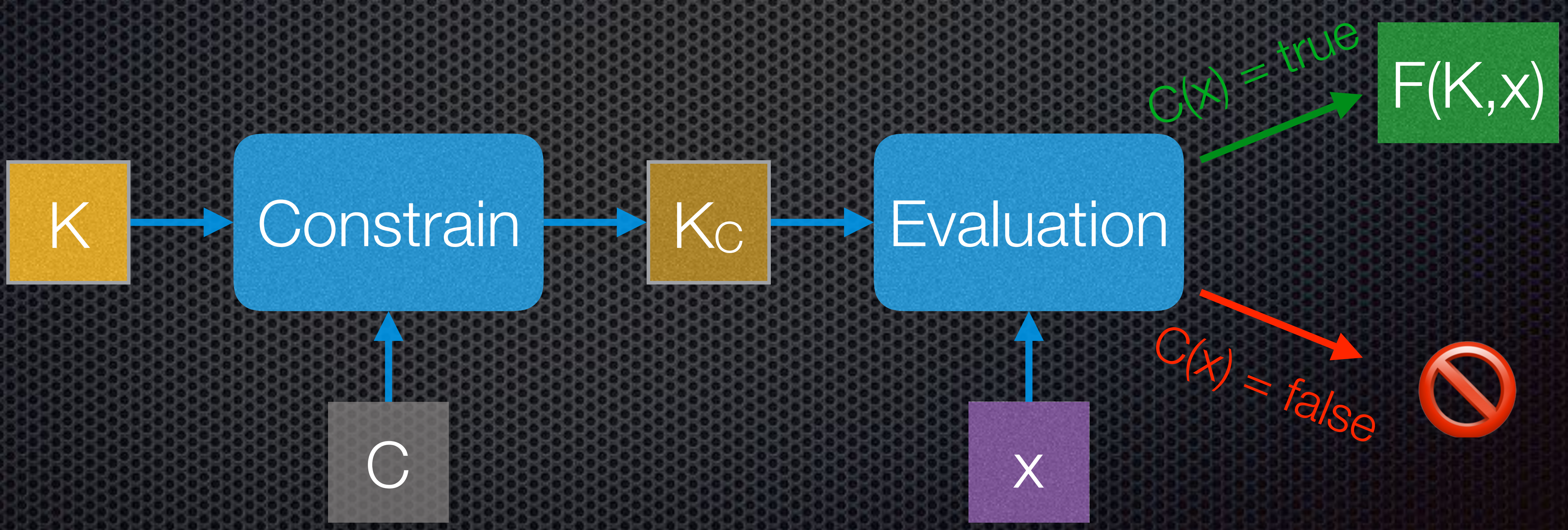
Constrained PRF

- Can we restrict the evaluation of $F(K_W,.)$ on $[1,n]$?



Constrained PRF

- Can we restrict the evaluation of $F(K_w,.)$ on $[1,n]$?

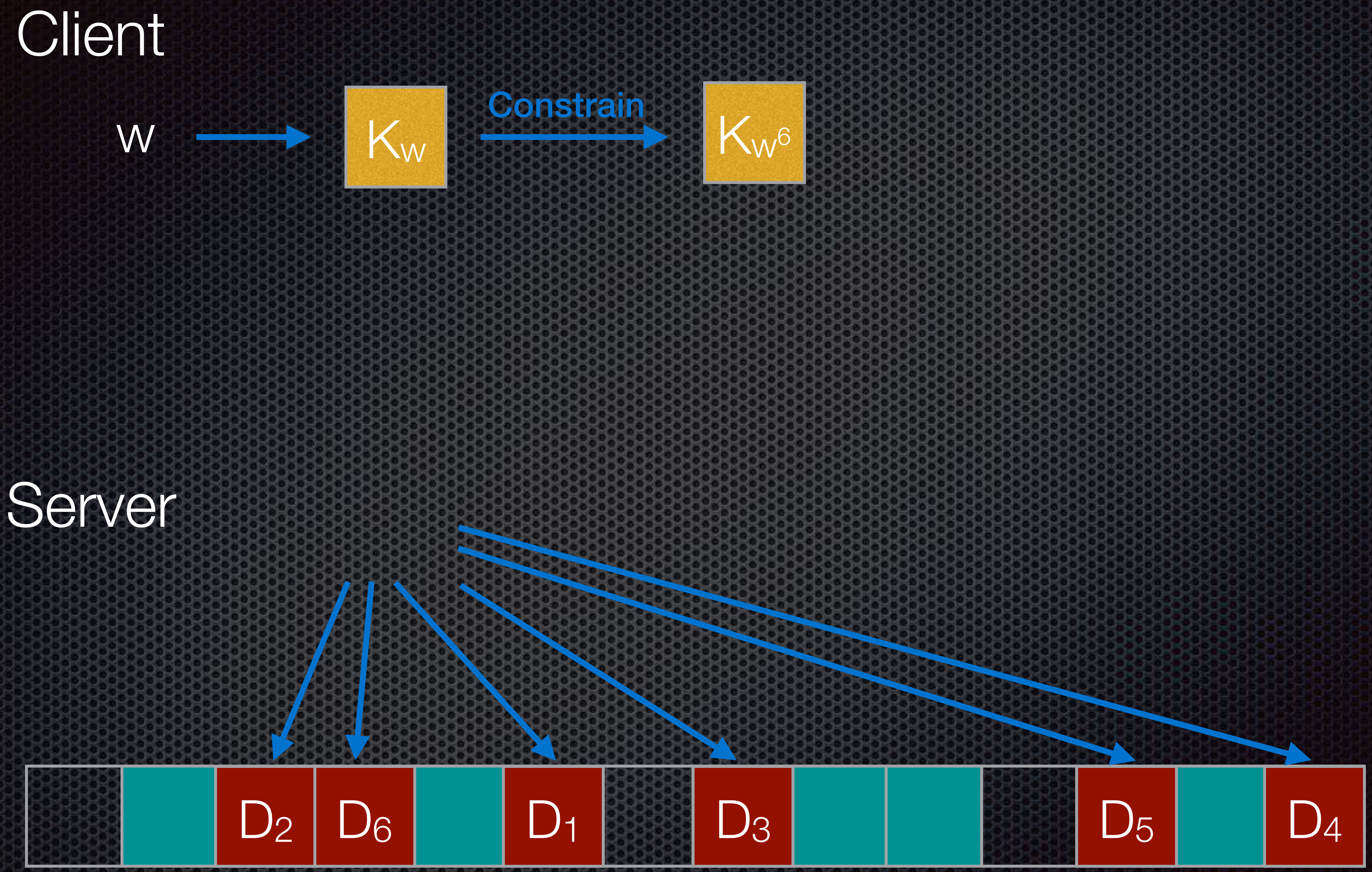


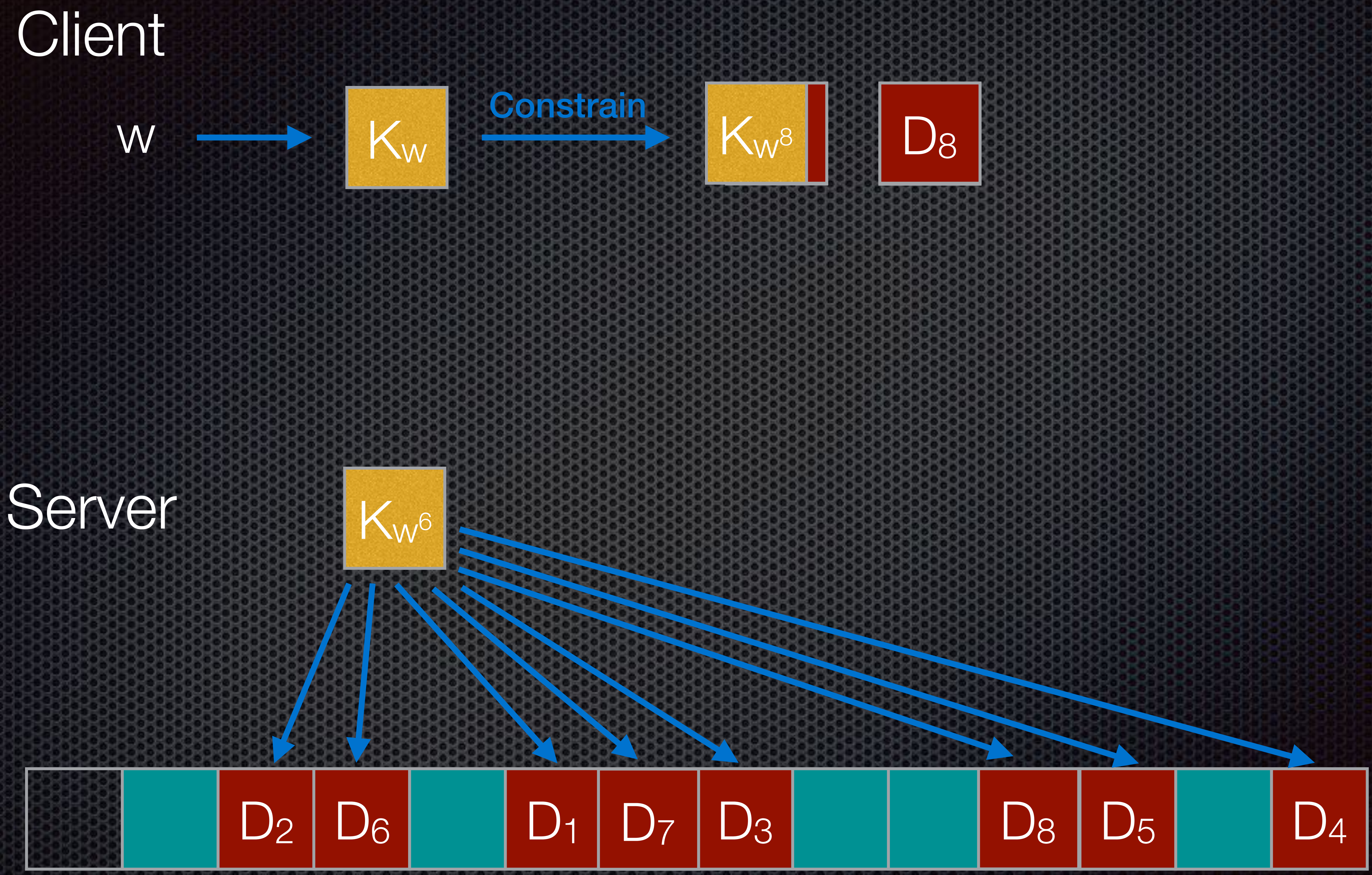
Range-Constrained PRF

- Consider the condition C_n:

  $C_n(x) = true$ if and only if $1 \leq x \leq n$ (range condition)

- $K^n = Constrain(K, C_n)$ can only be used to evaluate F on $[1,n]$


Client
w
K_w
Constrain
K_{w^6}
Server
D_2
D_6
D_1
D_3
D_5
D_4


Client
w
Kw
Constrain
Kw8
D8
Server
Kw6
D2
D6
D1
D7
D3
D8
D5
D4

Diana

- Instantiate the CPRF *F* with a tree-based PRF construction
- Asymptotically less efficient than Σοφος
- In practice, a lot better. Always IO bounded (for both searches and updates)
- Search: <1µs per match (on RAM)
  Update: 174 000 entries per second
  (4300 for Σοφος)

Can we do better?

- Similarly to the ORAM lower bound, we can show that the computational overhead of an update for a forward-private scheme is

$$\Omega\left(\frac{\log |W|}{\log \sigma}\right)$$

- Σοφος is optimal (constant-time update, $\sigma = |W|$)

Deletions

Deletions

How to delete entries in an encrypted database?

- Existing schemes use a 'revocation list'
- Pb: the deleted information is still revealed to the server
- Backward privacy: 'nothing' is leaked about the deleted documents

Backward privacy

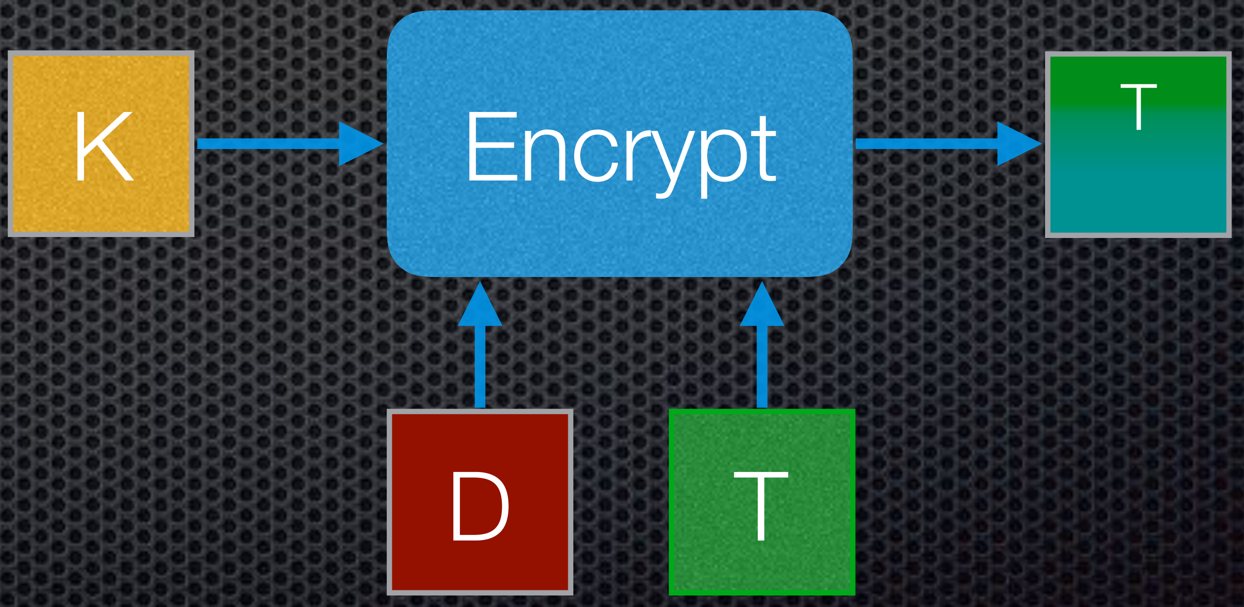
Baseline: the client fetches the encrypted lists of inserted and deleted documents, locally decrypts and retrieves the documents.

- ✓ Optimal security
- ✗ 2 interactions
- ✗ Complexity (communication & computation) :
  # insertions (vs. # results)

Backward privacy with optimal updates & comm.

Could we prevent the server from decrypting some entries?

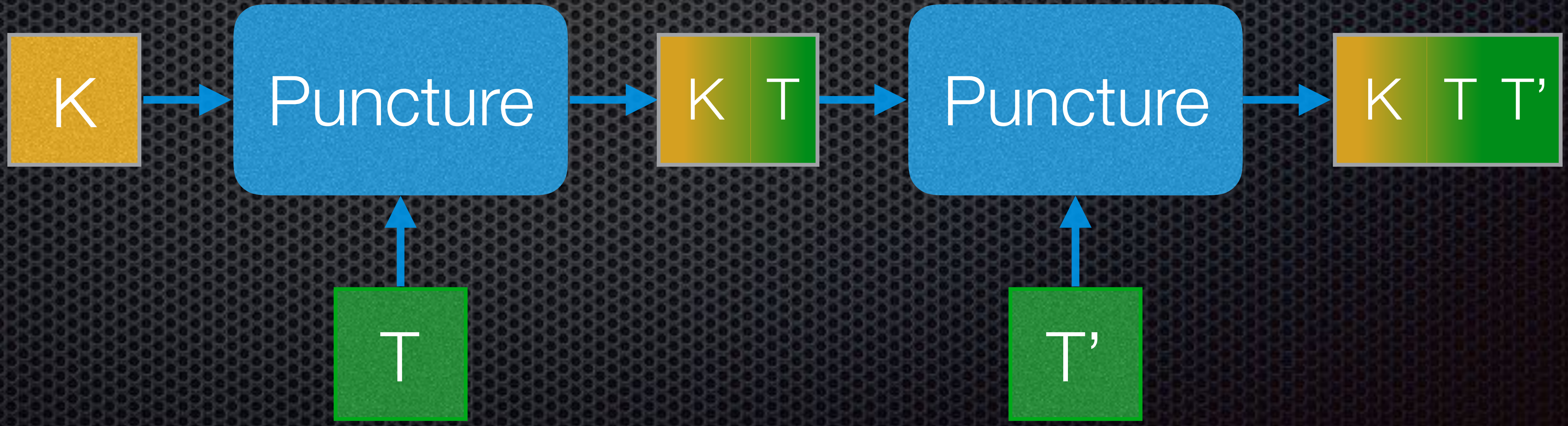
- Puncturable Encryption [GM'15]: Revocation of decryption capabilities for specific messages



Backward privacy with optimal updates & comm.

Could we prevent the server from decrypting some entries?

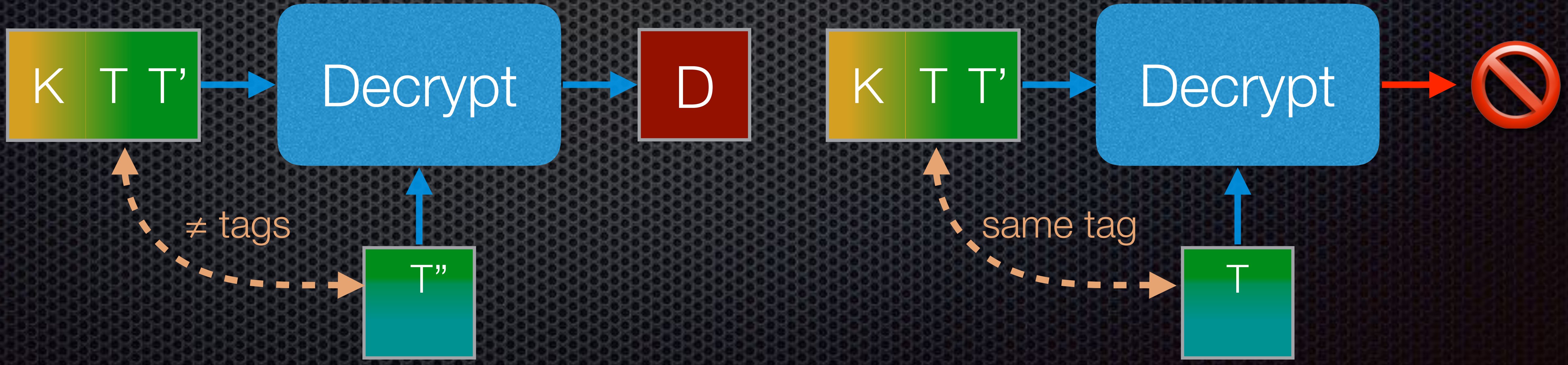
- Puncturable Encryption [GM'15]: Revocation of decryption capabilities for specific messages



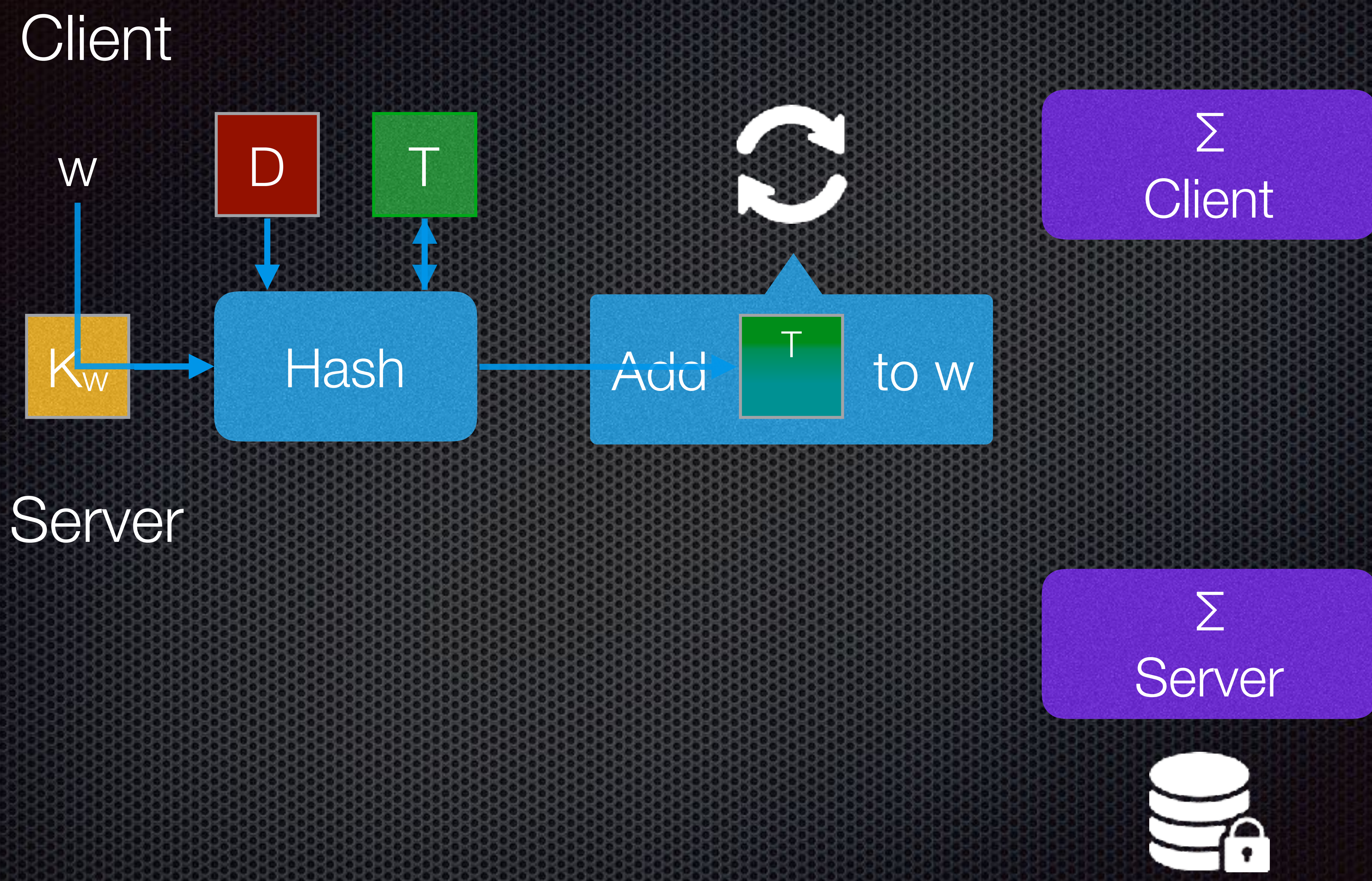
Backward privacy with optimal updates & comm.

Could we prevent the server from decrypting some entries?

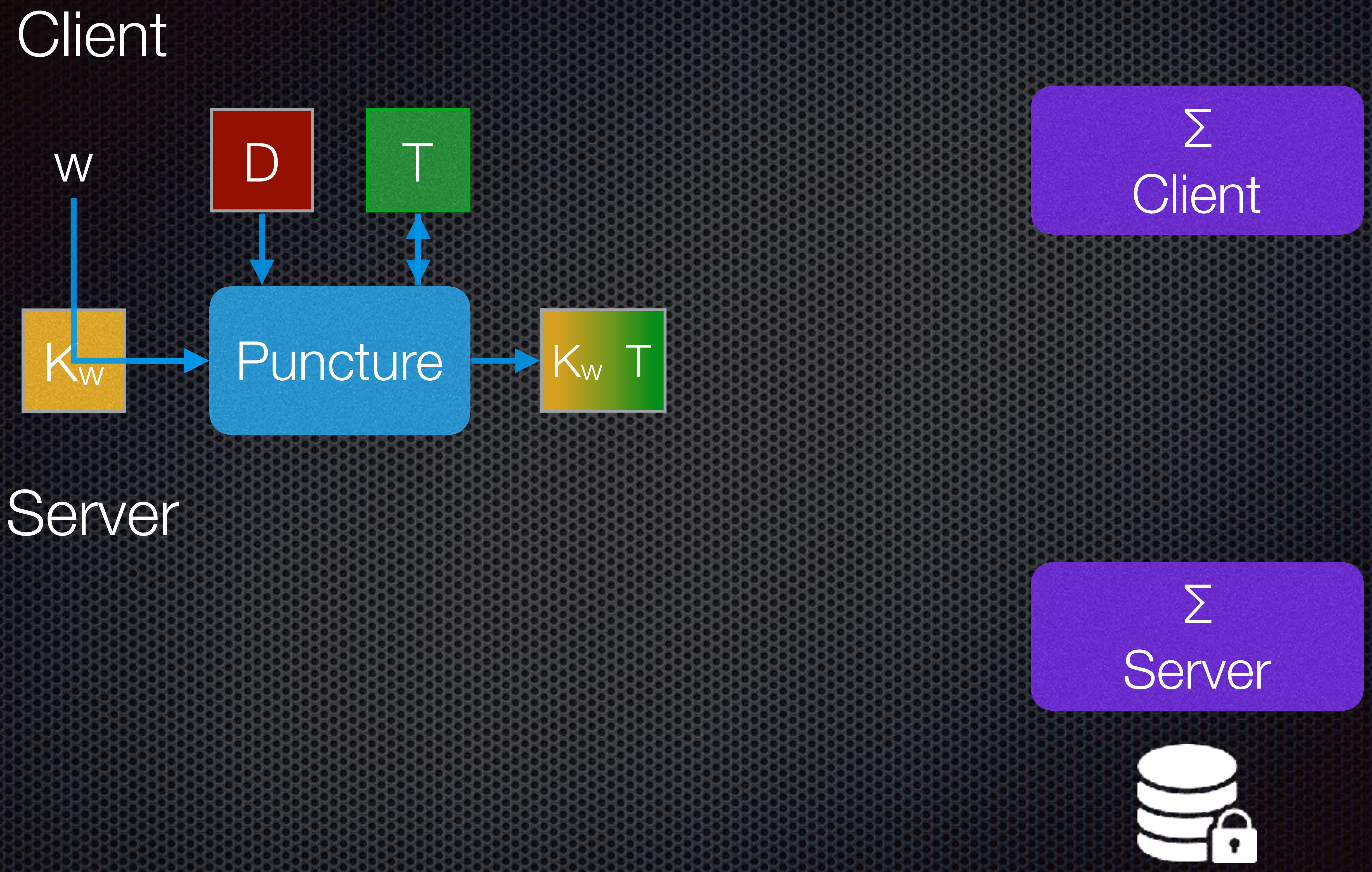
- Puncturable Encryption [GM'15]: Revocation of decryption capabilities for specific messages



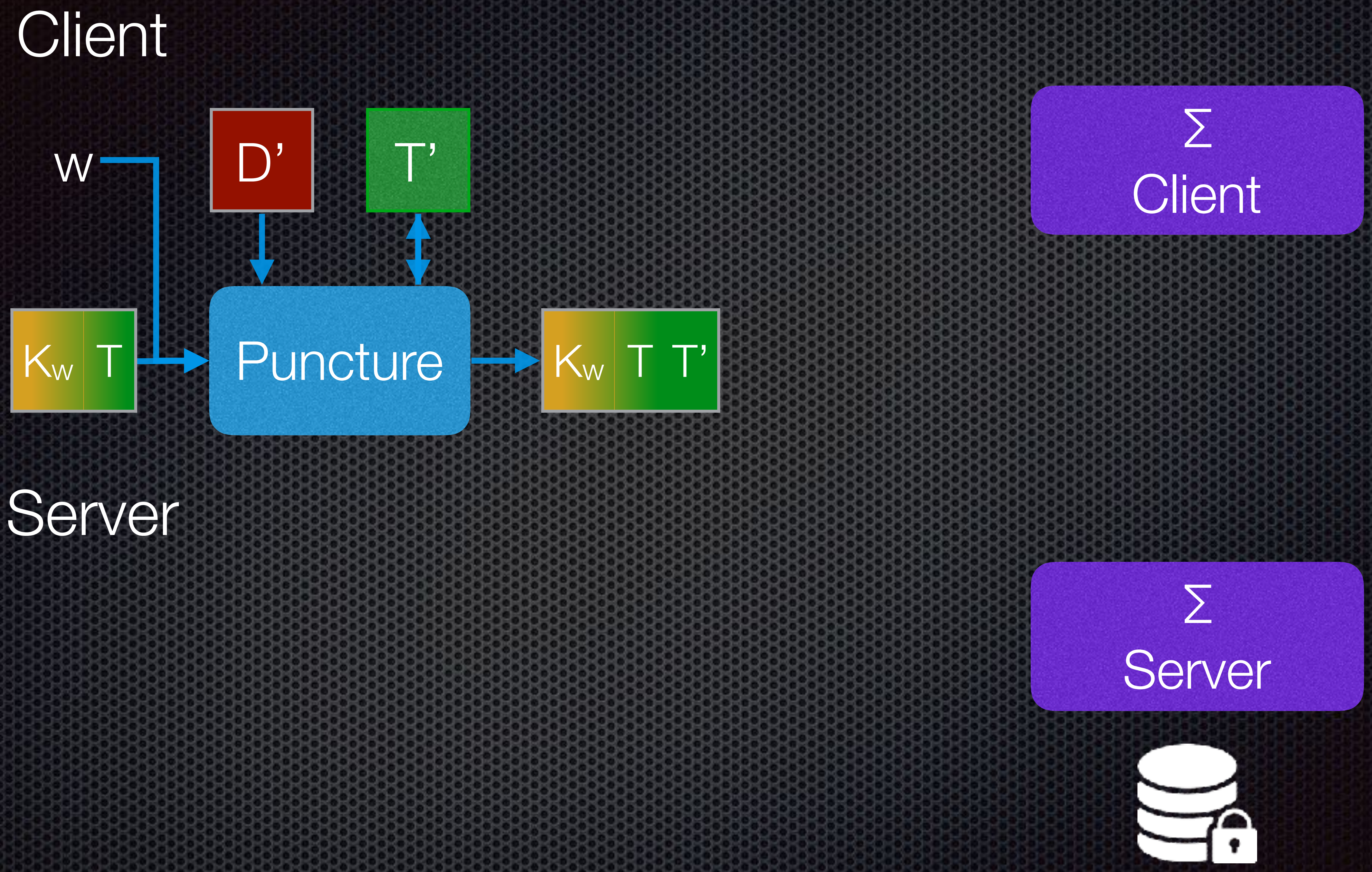
Insertion



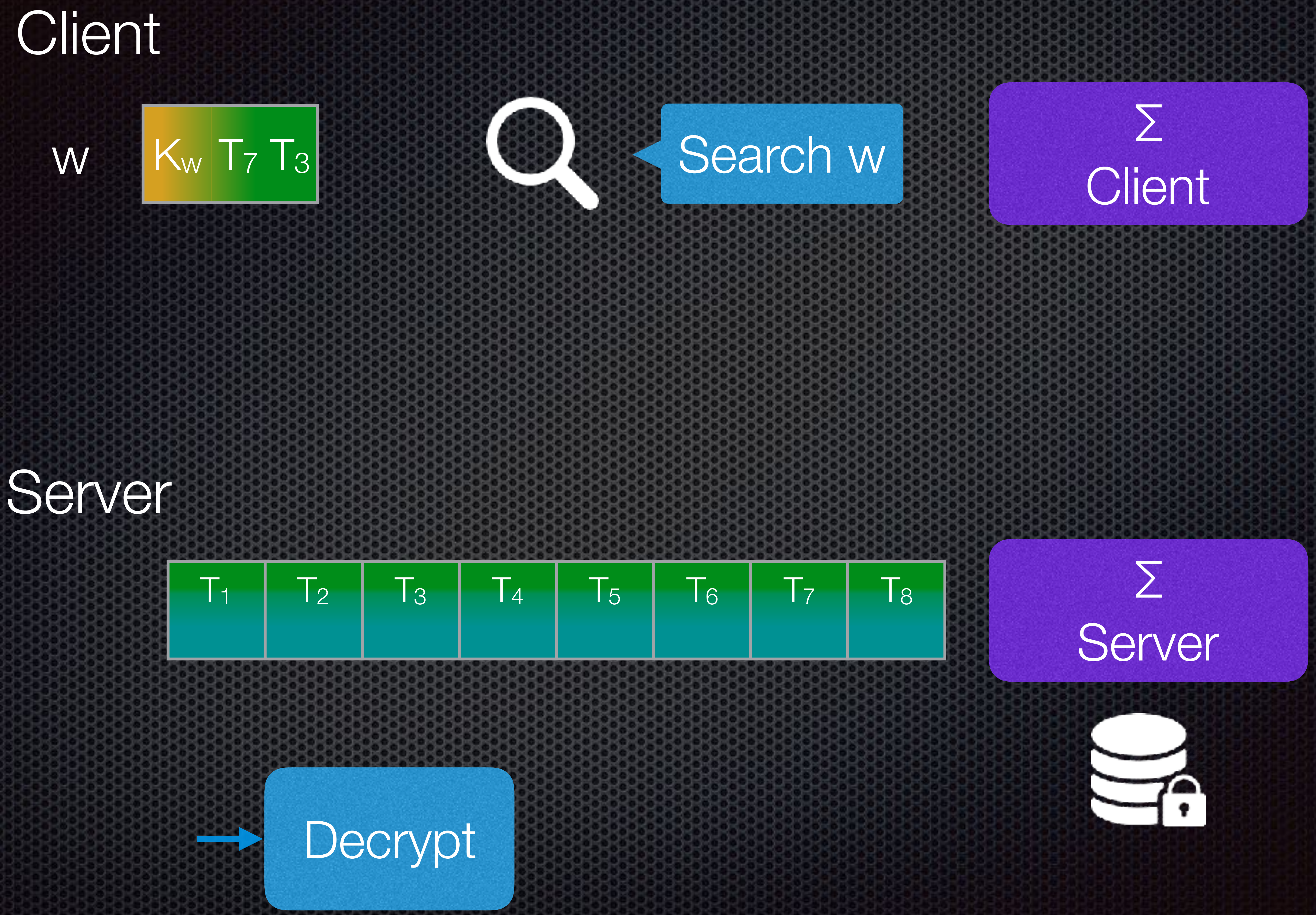
Deletion



Deletion



Search


Client
w
Kw T7 T3
Search w
Σ Client
Server
T1 T2 T3 T4 T5 T6 T7 T8
Σ Server
Decrypt

Janus

Good:

✓ Forward & backward-private

✓ Optimal update complexity

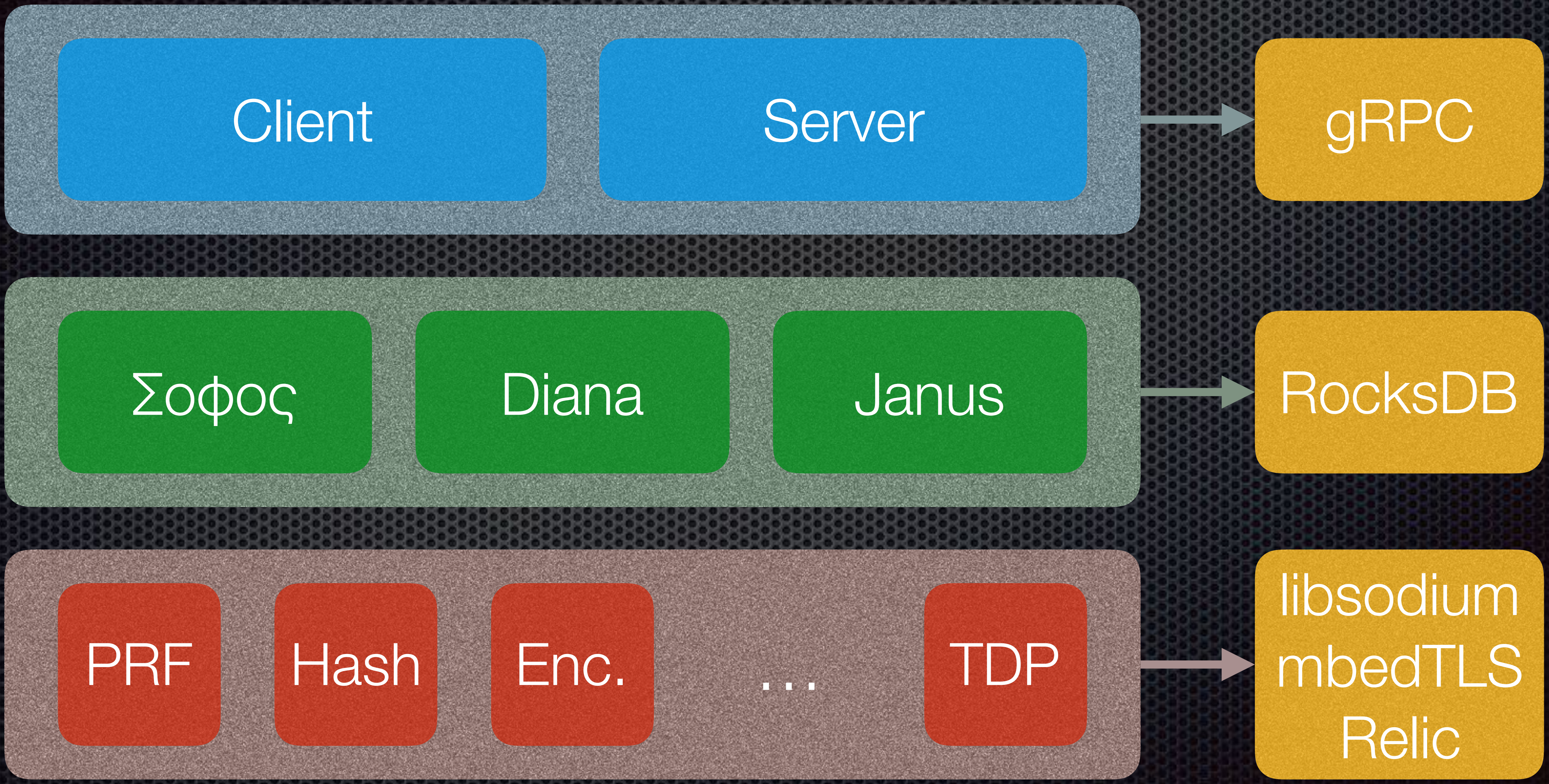
✓ Optimal communication

Not so good:

✗ $O(n_w . d_w)$ search comp.

✗ Uses pairings (not fast)

Implementation of SE



OpenSSE

- Goal: fast and secure implementation of SE schemes
- 10 700 C/C++ LoC (crypto: 6500, schemes: 4200)
- Open Source: opensse.github.io
- And its documented !!! (at least for the crypto)

Other works on searchable encryption

- Verifiable SSE: check that the results returned by the server are correct. Constructions and lower bounds
- Analysis of recent attacks (leakage-abuse attacks) that only use the leakage to break the security of schemes. Proposed countermeasures.

Conclusion

- Forward privacy
  - Updates do not leak information about the past events
  - Two efficient constructions Σοφος and Diana
- Backward privacy
  - Deletions are not recoverable by the server
  - Janus: backward privacy with optimal communication

Conclusion

- SE involves very diverse topics: theoretical CS, cryptanalysis, cryptographic primitives, systems, …
- Real world cryptography, with great impact

Publications

Searchable Encryption:

- [B Fouque Pointcheval - *ePrint 16*]: Verifiable Dynamic Symmetric Searchable Encryption: Optimality and Forward Security
- [B - *CCS 16*]: Σοφος: Forward Secure Searchable Encryption
- [B Minaud Ohrimenko - *CCS 17*]: Forward and Backward Private Searchable Encryption from Constrained Cryptographic Primitives
- [B Fouque - *ePrint 17*]: Thwarting Leakage Abuse Attacks against Searchable Encryption – A Formal Approach and Applications to Database Padding

Other:

- [B Popa Tu Goldwasser - *NDSS 15*]: Machine Learning Classification over Encrypted Data.
- [B Sanders - *AsiaCrypt 16*]: Trick or Tweak: On the (In)security of OTR's Tweaks

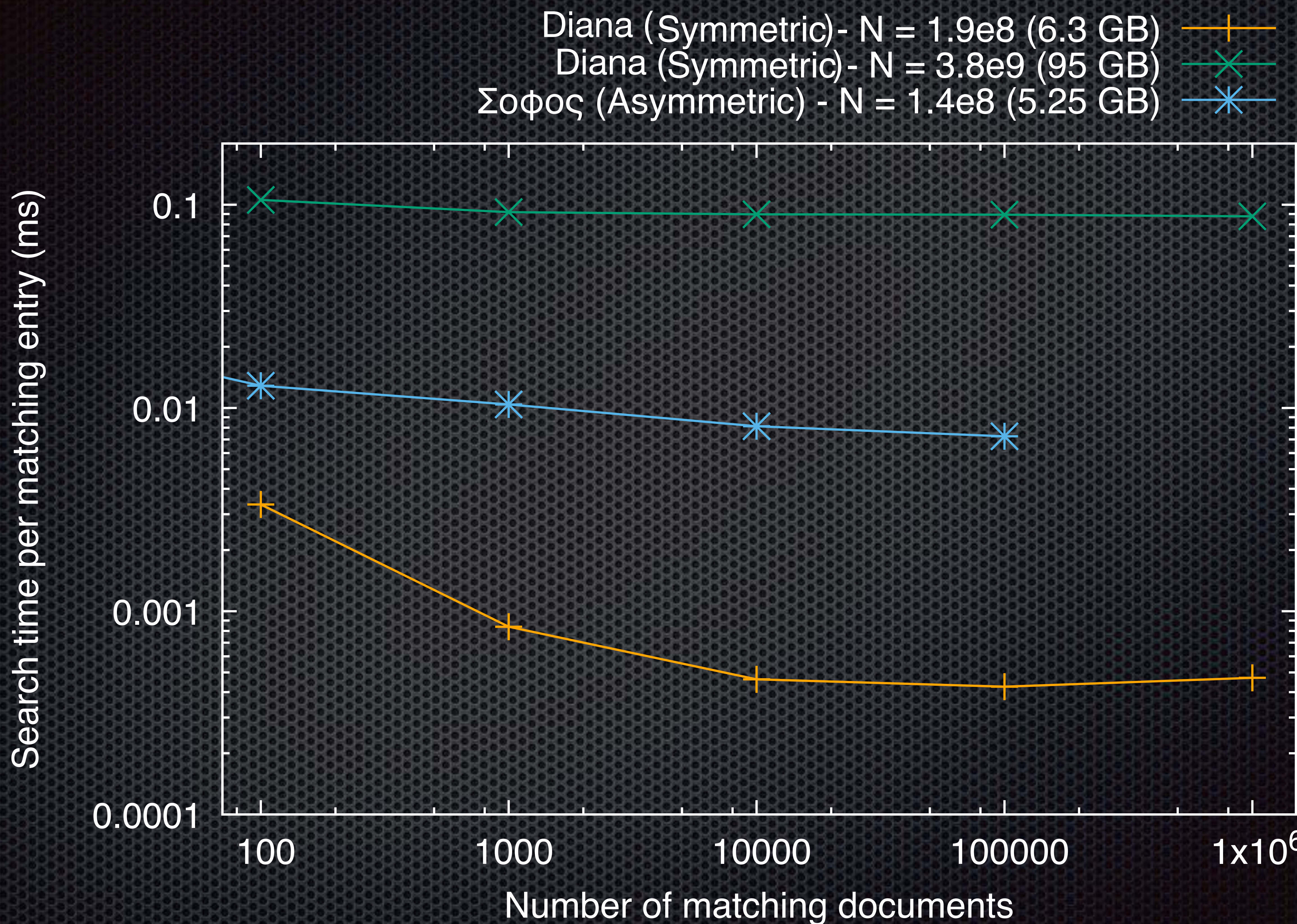
Verifiable SE

- The server might be malicious: return fake results, delete real results, …
- The client needs to verify the results

Verifiable SE

This is not free: lower bound (derived from [DNRV'09])

- If client storage is less than $|W|^{1-\varepsilon}$, search complexity has to be larger than $\log |W|$
- The lower bound is tight: using Merkle hash trees and set hash functions
- Many possible tradeoffs between search & update complexities



Diana (Symmetric)- N = 1.9e8 (6.3 GB)
Diana (Symmetric)- N = 3.8e9 (95 GB)
Σοφος (Asymmetric) - N = 1.4e8 (5.25 GB)
Search time per matching entry (ms)
0.1
0.01
0.001
0.0001
100
1000
10000
100000
1x10^6
Number of matching documents

Crypto vs. Seek time

The magic world of searchable encryption:

- Symmetric crypto is free
- Asymmetric crypto is not overly expensive
- A lot of the cost comes from the non-locality of memory accesses

Locality vs. Caching

- The OS is ‘smart’: it caches memory.
- Be careful when you are testing your construction on small databases
- Once the database is cached, non locality disappears
- Beware of the evaluation of performance

Evaluating the security

- Use the leakage function from the security definitions
  - ✓ Provable security
  - ✗ Very hard to understand the extend of the leakage
- Rely on cryptanalysis: leakage-abuse attacks
  - ✗ Maybe not the best adversary
  - ✓ ‘Real world’ implications

Evaluating the security

- State-of-the-art schemes leak the number of results of a query
  - ➡ Enough to recover the queries when the adversary knows the database [CGPR'15]
  - ➡ Counter-measure: padding (it has a cost)