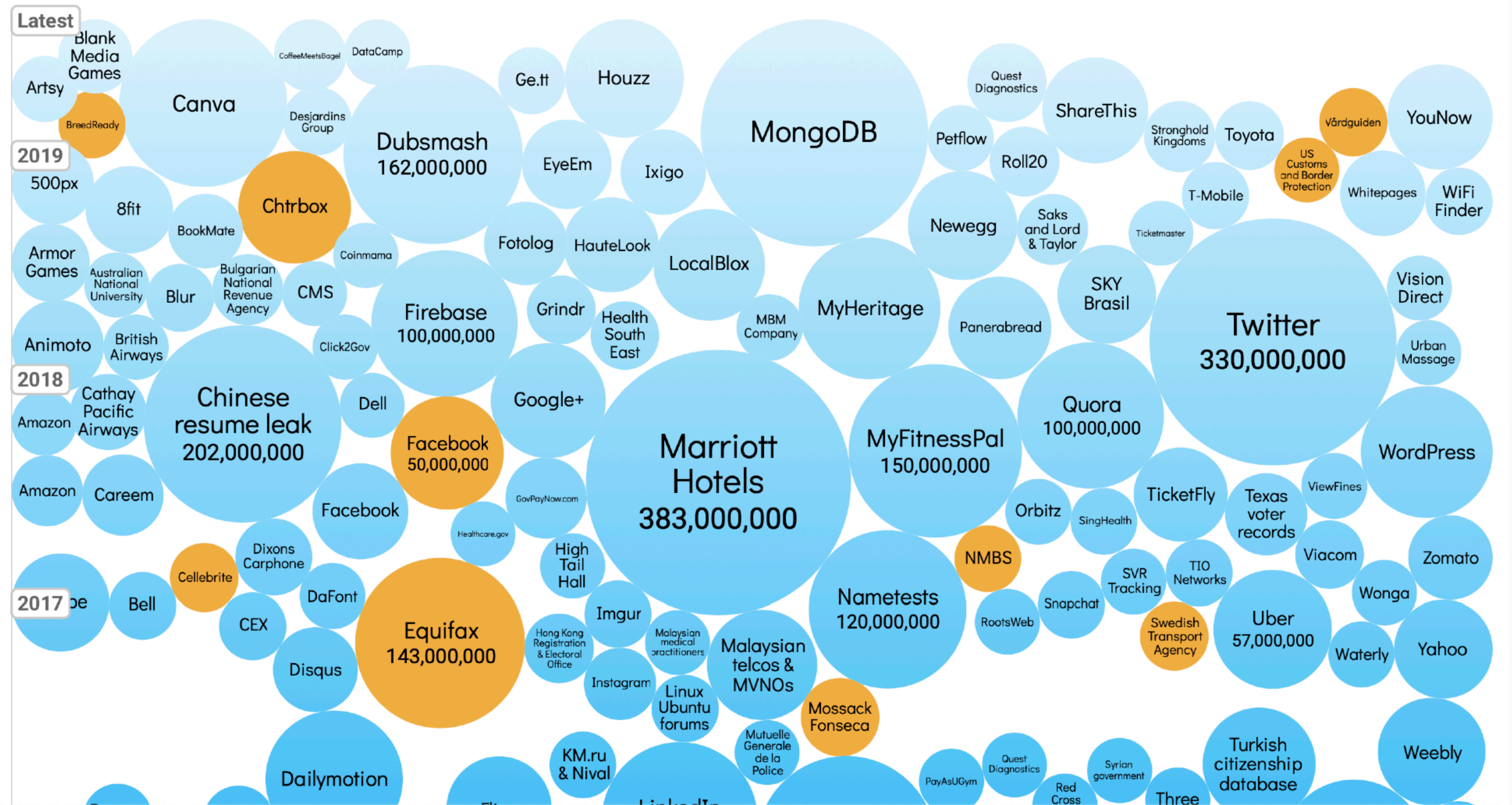
SAC Summer School 2019

Encrypted Search: Intro & Basics

Seny Kamara

BROWN

ENCRYPTED SYSTEMS LAB


Latest
2019
2018
2017
Blank Media Games
Artsy
BreedReady
Canva
CoffeeMeetsBagel
DataCamp
Desjardins Group
Ge.tt
Houzz
Dubsmash
162,000,000
MongoDB
Quest Diagnostics
ShareThis
Petflow
Roll20
Stronghold Kingdoms
Toyota
Vårdguiden
YouNow
US Customs and Border Protection
Whitepages
WiFi Finder
T-Mobile
500px
8fit
Chtrbox
EyeEm
Ixigo
Newegg
Saks and Lord & Taylor
Ticketmaster
BookMate
Coinmama
Fotolog
HauteLook
LocalBlox
Armor Games
Australian National University
Blur
Bulgarian National Revenue Agency
CMS
Firebase
100,000,000
Grindr
Health South East
MBM Company
MyHeritage
Panerabread
SKY Brasil
Twitter
330,000,000
Vision Direct
Urban Massage
Animoto
British Airways
Click2Gov
Chinese resume leak
202,000,000
Dell
Google+
Marriott Hotels
383,000,000
MyFitnessPal
150,000,000
Quora
100,000,000
WordPress
Cathay Pacific Airways
Amazon
Facebook
50,000,000
GovPayNow.com
Amazon
Careem
Facebook
Healthcare.gov
Orbitz
SingHealth
TicketFly
Texas voter records
ViewFines
Dixons Carphone
Cellebrite
High Tail Hall
NMBS
TIO Networks
Viacom
Zomato
Bell
DaFont
CEX
Equifax
143,000,000
Imgur
Hong Kong Registration & Electoral Office
Malaysian medical practitioners
Nametests
120,000,000
RootsWeb
Snapchat
SVR Tracking
Uber
57,000,000
Wonga
Disqus
Instagram
Malaysian telcos & MVNOs
Swedish Transport Agency
Waterly
Yahoo
Linux Ubuntu forums
Mossack Fonseca
Dailymotion
KM.ru & Nival
Mutuelle Generale de la Police
Quest Diagnostics
PayAsUGym
Syrian government
Red Cross
Turkish citizenship database
Three
Weebly

14,717,618,286*

4%

* since 2013

Why so Few?

Incompetence?

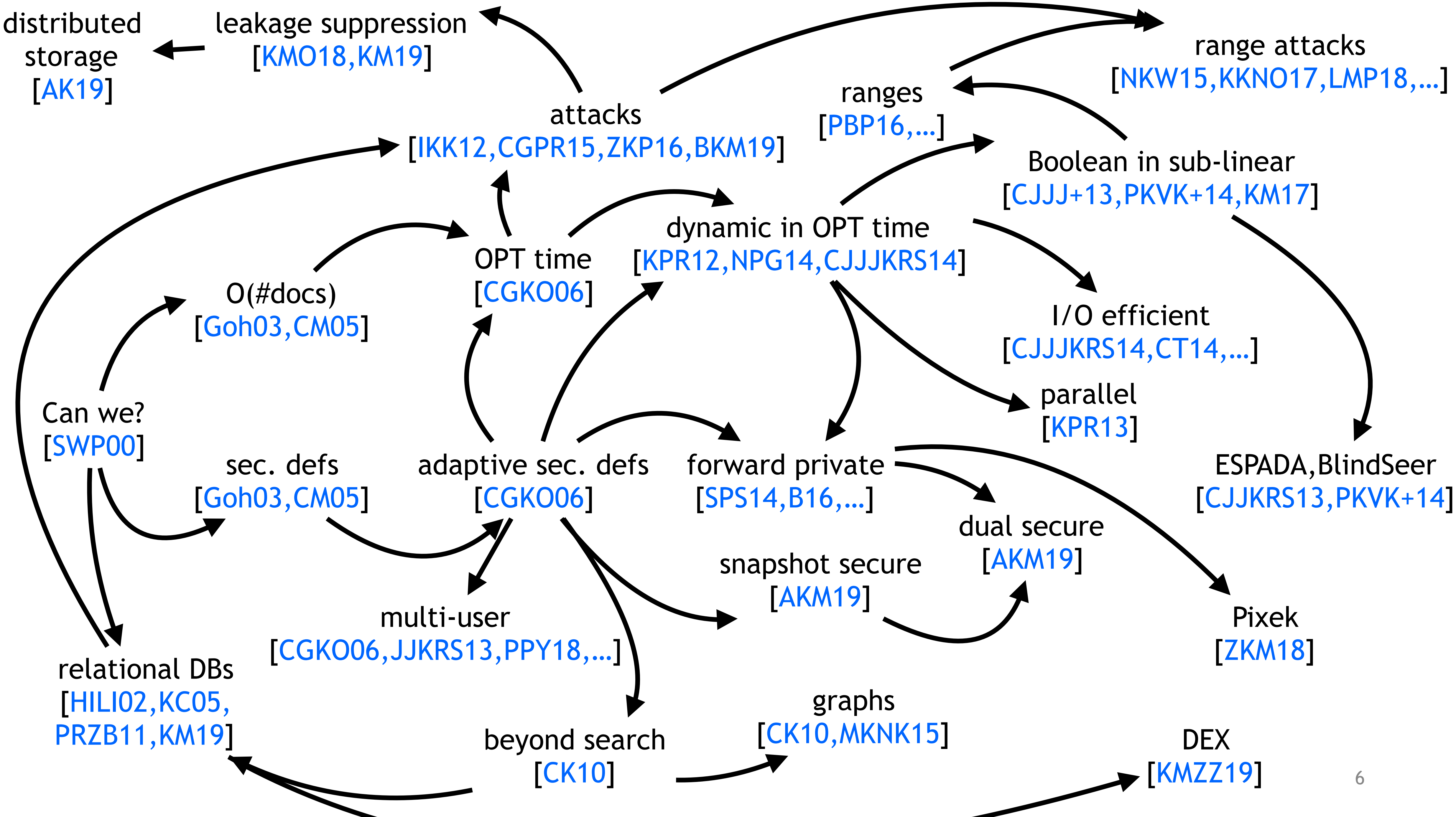
Lazyness?

Cost?

"…because it would have hurt Yahoo's ability to index and search message data…"

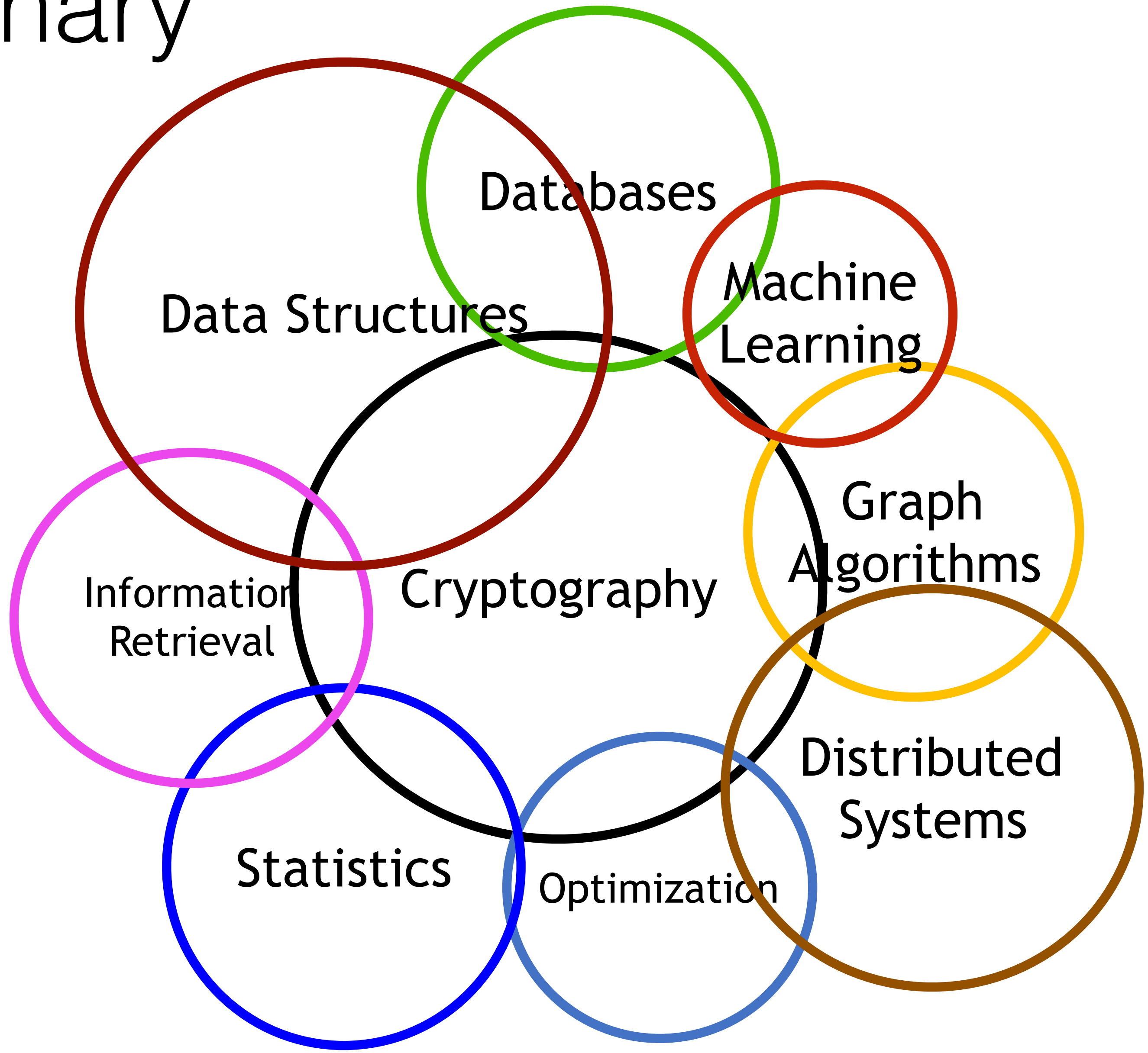
— J. Bonforte in NY Times

Q: can we search on encrypted data?



distributed storage [AK19]
leakage suppression [KMO18,KM19]
range attacks [NKW15,KKNO17,LMP18,…]
ranges [PBP16,…]
attacks [IKK12,CGPR15,ZKP16,BKM19]
Boolean in sub-linear [CJJJ+13,PKVK+14,KM17]
dynamic in OPT time [KPR12,NPG14,CJJJKRS14]
OPT time [CGKO06]
O(#docs) [Goh03,CM05]
I/O efficient [CJJJKRS14,CT14,…]
parallel [KPR13]
Can we? [SWP00]
sec. defs [Goh03,CM05]
adaptive sec. defs [CGKO06]
forward private [SPS14,B16,…]
ESPADA,BlindSeer [CJJKRS13,PKVK+14]
dual secure [AKM19]
snapshot secure [AKM19]
multi-user [CGKO06,JJKRS13,PPY18,…]
Pixek [ZKM18]
relational DBs [HILI02,KC05, PRZB11,KM19]
graphs [CK10,MKNK15]
beyond search [CK10]
DEX [KMZZ19]

Interdisciplinary



Real-World Problem

- Major companies
  - Microsoft, SAP
  - MongoDB, Cisco
  - Google Research
  - Hitachi, Fujitsu
  - more…
- Funding agencies
  - NSF
  - IARPA
  - DARPA
- Startups
  - Ciphercloud
  - Skyhigh Networks
  - Bitglass
  - Baffle
  - Cossack Labs
  - Strong Salt, Overnest
  - many many more

Encrypted Search (Building Blocks)

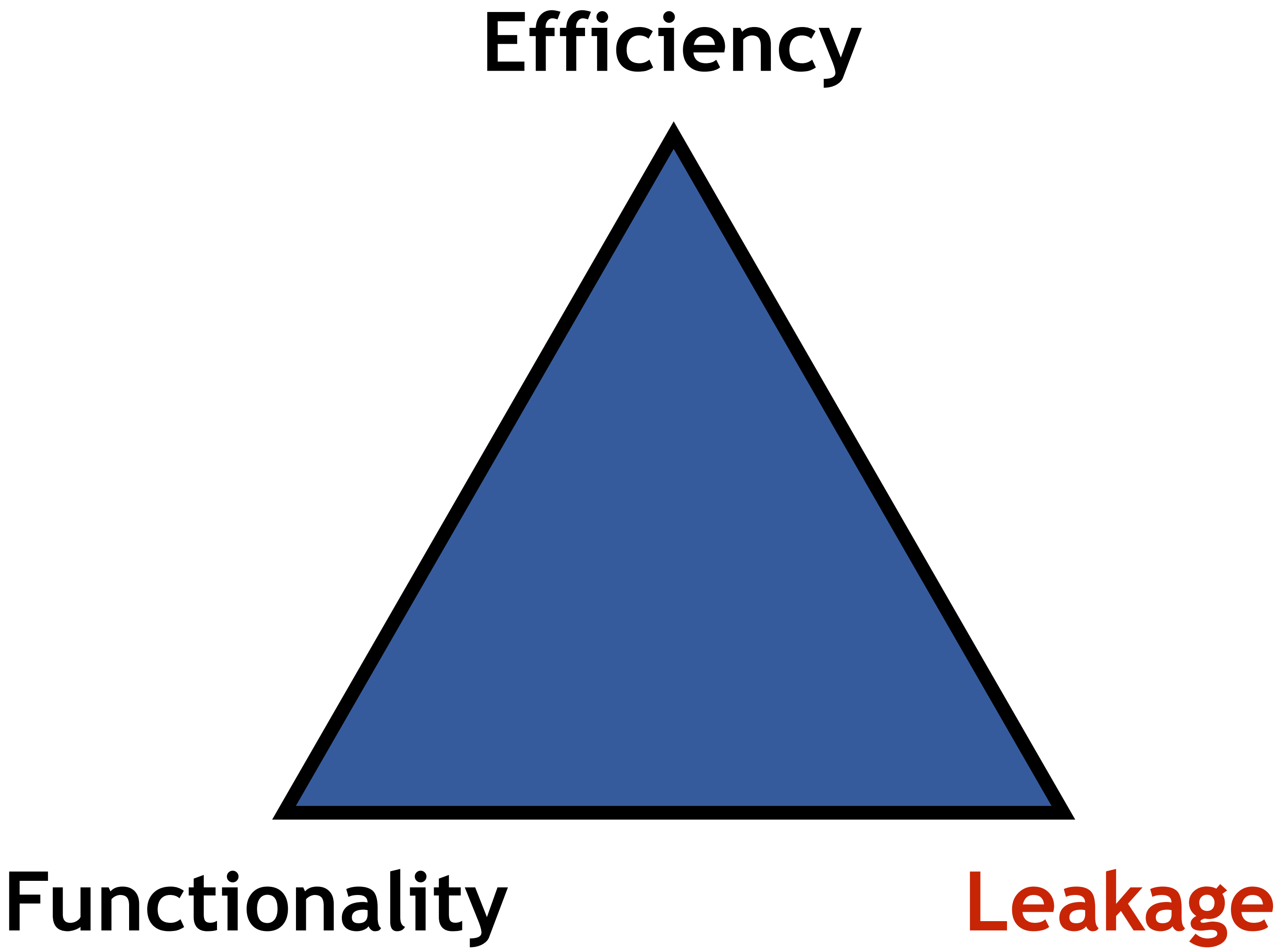
Property-Preserving Encryption (PPE)

Functional Encryption

Structured Encryption (STE)

Fully-Homomorphic Encryption (FHE)

Oblivious RAM (ORAM)


Efficiency
Functionality
Leakage

What is Search?

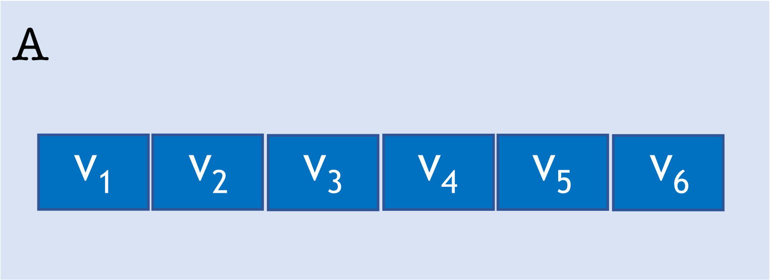
- Complexity regimes
  - linear search: **O(n)**
  - sub-linear search: **o(n)**
- Algorithmic paradigms
  - with pre-processing
  - without pre-processing
- For medium to large data
  - sub-linear search is a *requirement*; not an option

	Without Pre-Processing	**With Pre-Processing**
Linear	sequential scan	not interesting
Sub-Linear	read sub-set of input (*errors*)	data structures

Background: Data Structures

- Abstract data types
  - capture functionality
  - ex: dictionary
- Data structures
  - instantiate ADTs
  - ex: hash table, binary search tree
- As common in CS
  - we sometimes blur the distinction

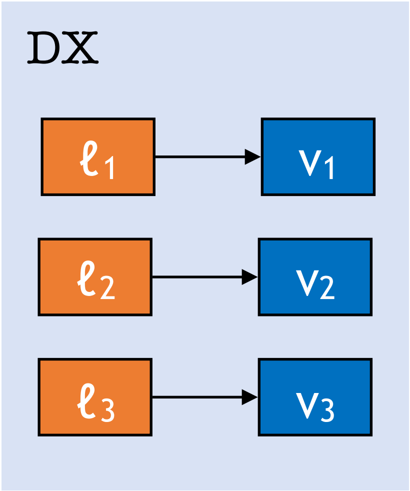
- Arrays store values



- Write: **A[i] :=** v_i
- Read: **A[i]** returns v_i

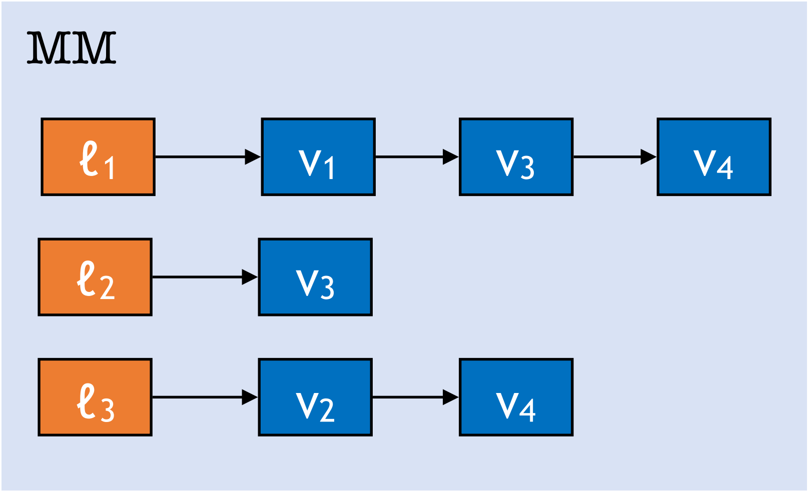
Background: Data Structures

- Dictionaries map labels to values



- Put: **DX[ℓ_2] := v_2**
- Get: **DX[ℓ_2]** returns v_2

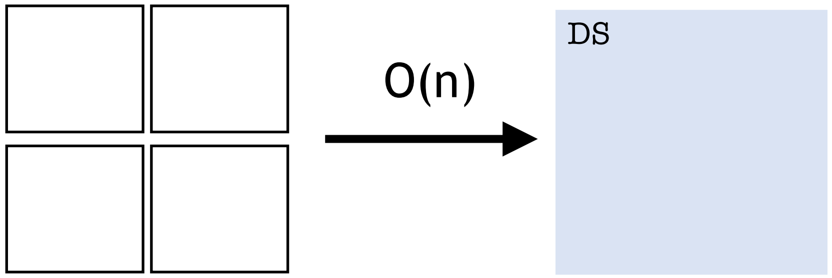
- Multi-Maps map labels to tuples



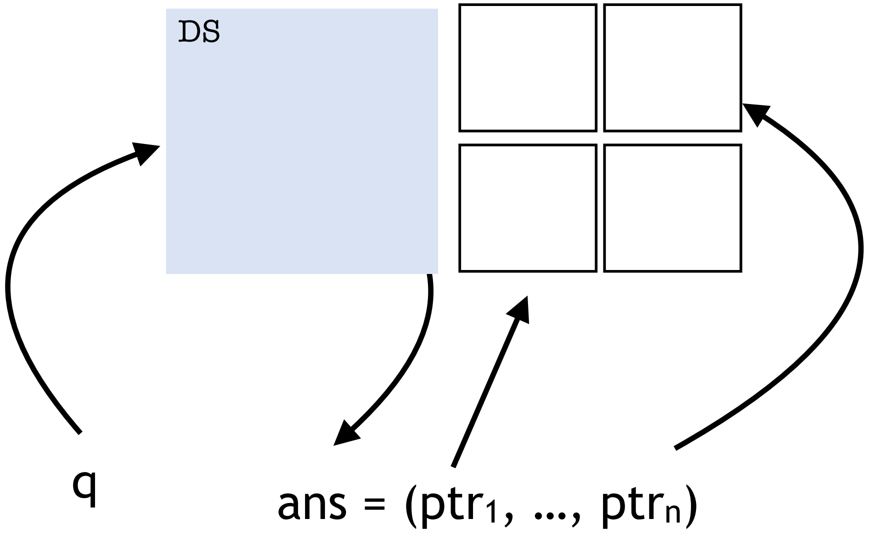
- Put: **MM[ℓ_3]:= (v_2,v_4)**
- Get: **MM[ℓ_3]** returns **(v_2,v_4)**

Keyword Search in Sub-Linear Time

<u>Setup time</u>

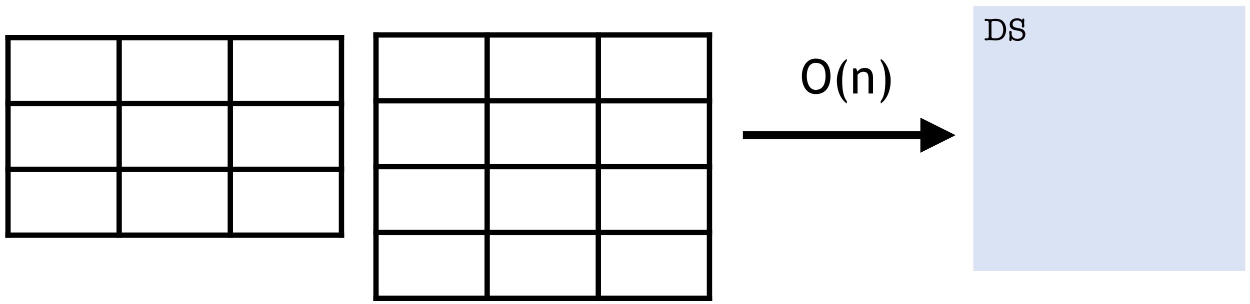


<u>Query time</u>

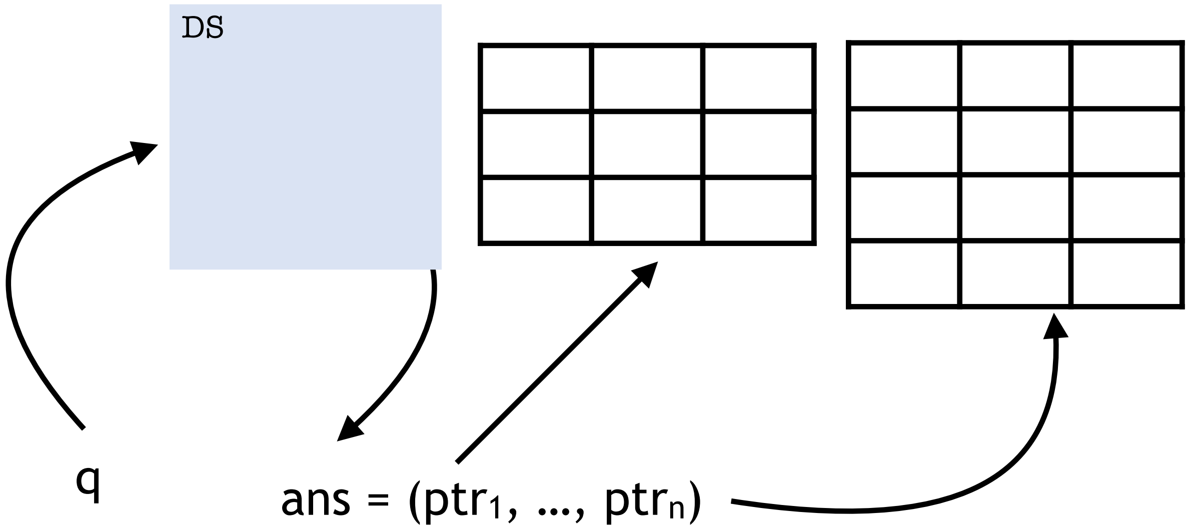


Database Queries in Sub-Linear Time

Setup time



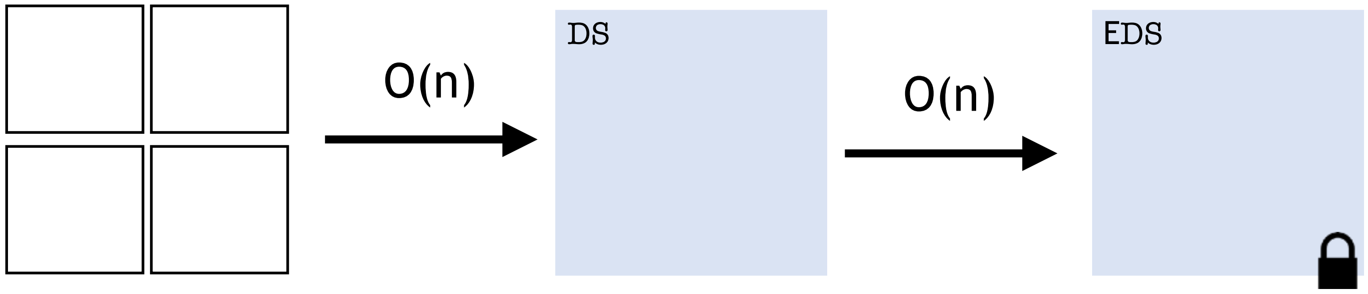
Query time



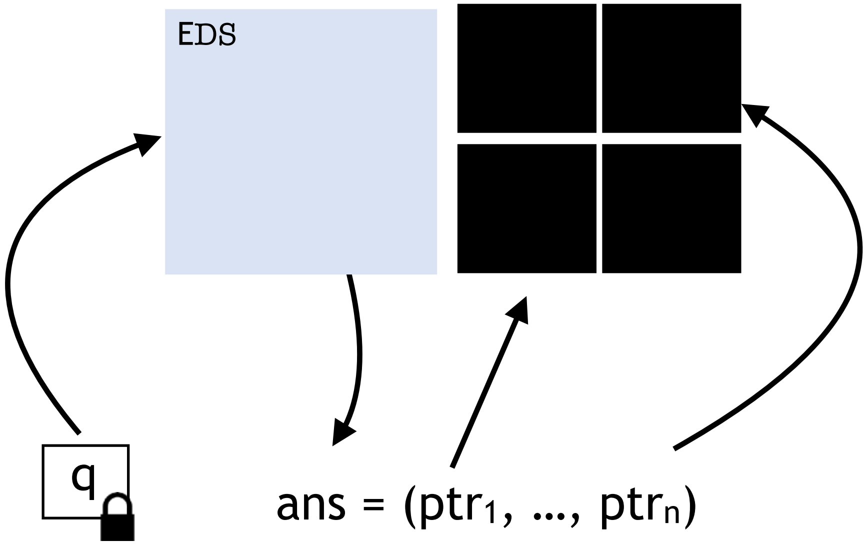
Q: how do we do sub-linear search on encrypted data?

Encrypted Keyword Search in *Sub-Linear* Time

Setup time

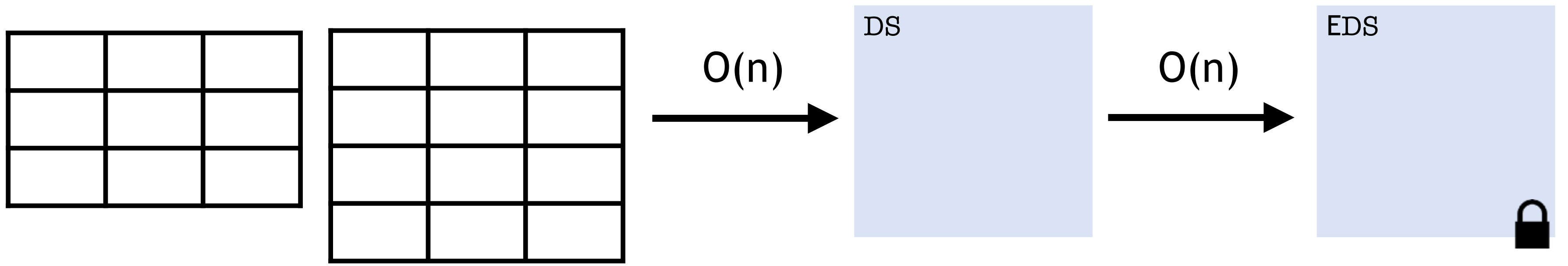


Query time

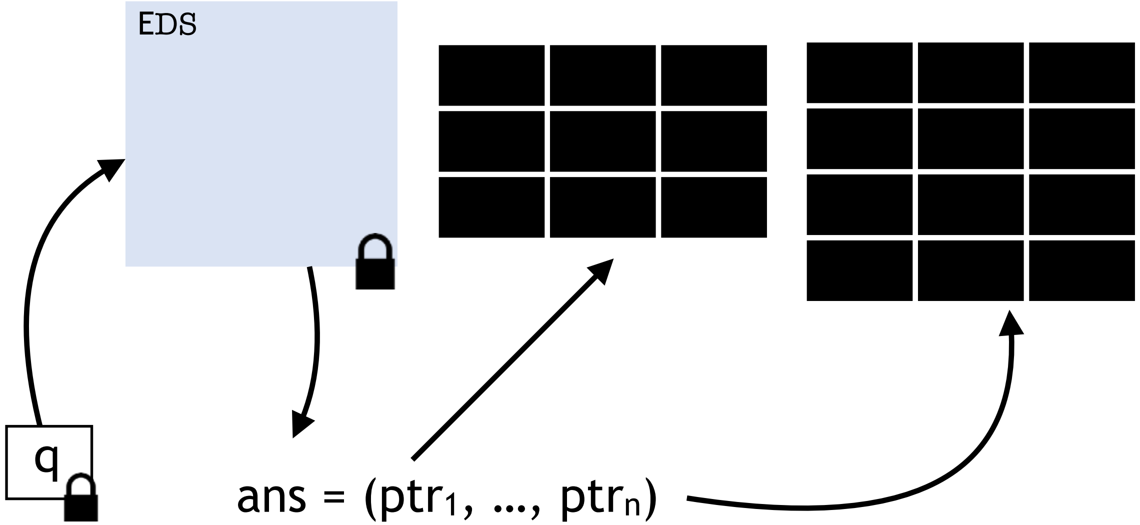


Encrypted Database Queries in Sub-Linear Time

Setup time



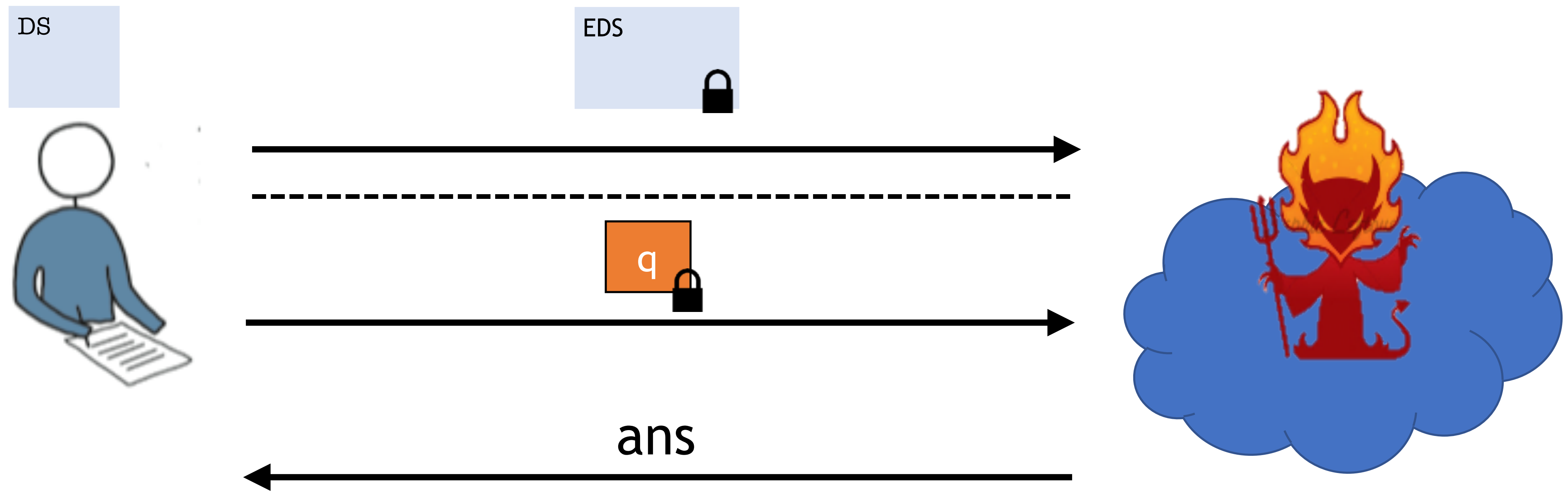
Query time



Q: how do we formalize *encrypted* data structures?

Structured Encryption

[Chase-K.10]

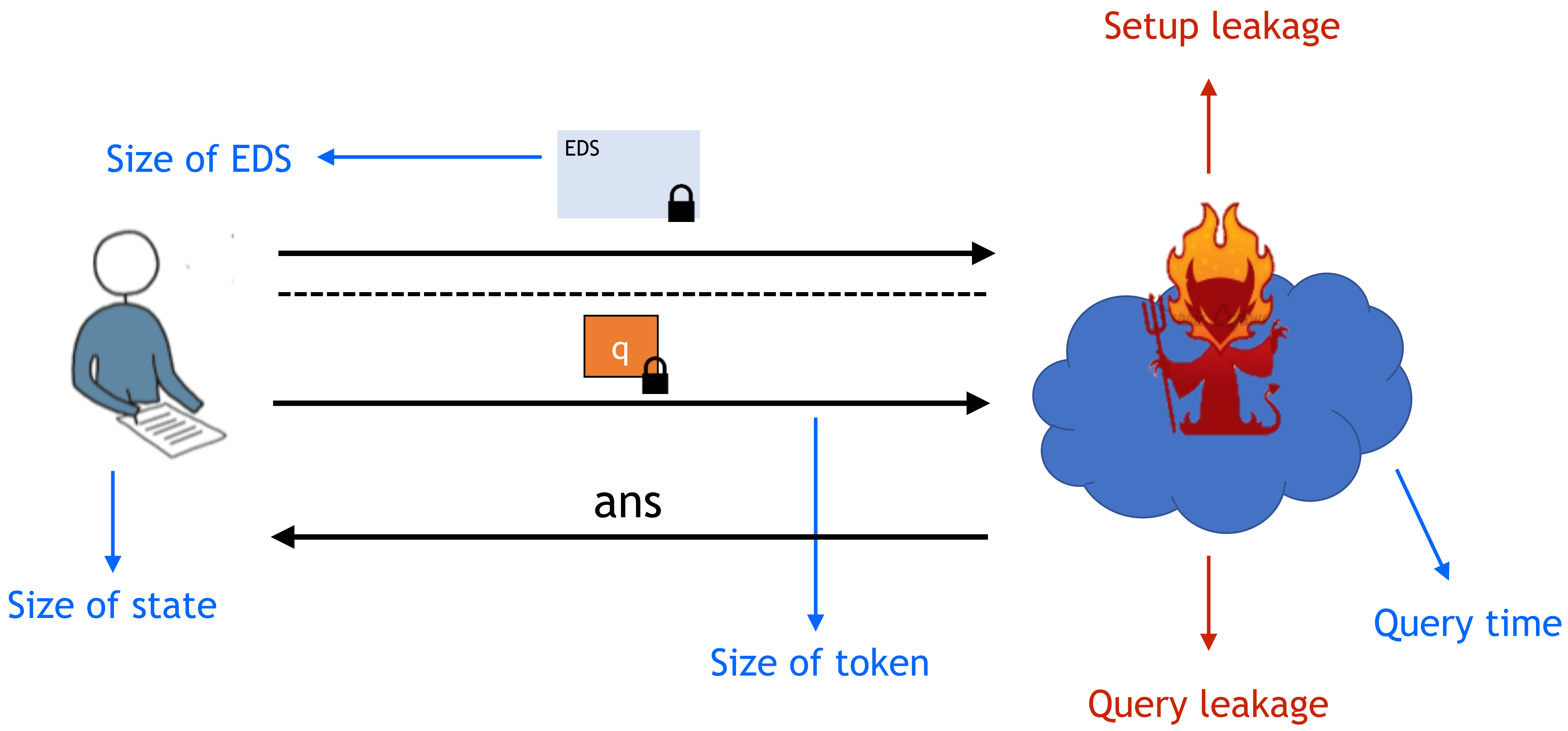


Setup(1^k, DS) $\longrightarrow$ (K, EDS)

Query(EDS, tk) $\longrightarrow$ ans

Token(K, q) $\longrightarrow$ tk

Desiderata

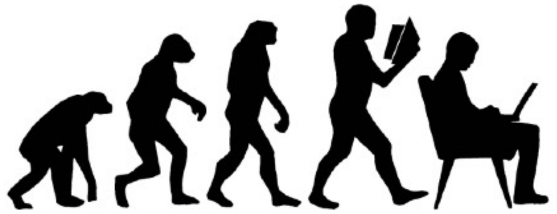


Setup leakage
Size of EDS
EDS
q
ans
Size of state
Size of token
Query time
Query leakage

Structured Encryption
[Chase-K.10]

- Many variants of STE
  - response-revealing
    - EDS query reveals answer in plaintext
  - response-hiding
    - EDS query reveals encrypted answer
  - non-interactive queries
    - clients sends single message called a token
  - interactive queries
    - client and server execute multi-round protocol

Evolution of Structured Encryption



Efficiency

- ‘00 Linear in file length [SWP00]
- ‘03 Linear in #docs [Goh03]
- ‘06 Optimal [CGKO06,CK10]
- ‘12 Optimal Dynamic [KPR12,CJJJKRS14]
- ‘14 I/O efficient [CT14,CJJJKRS14,ANSS16,DPP18],ASS18]

Expressiveness

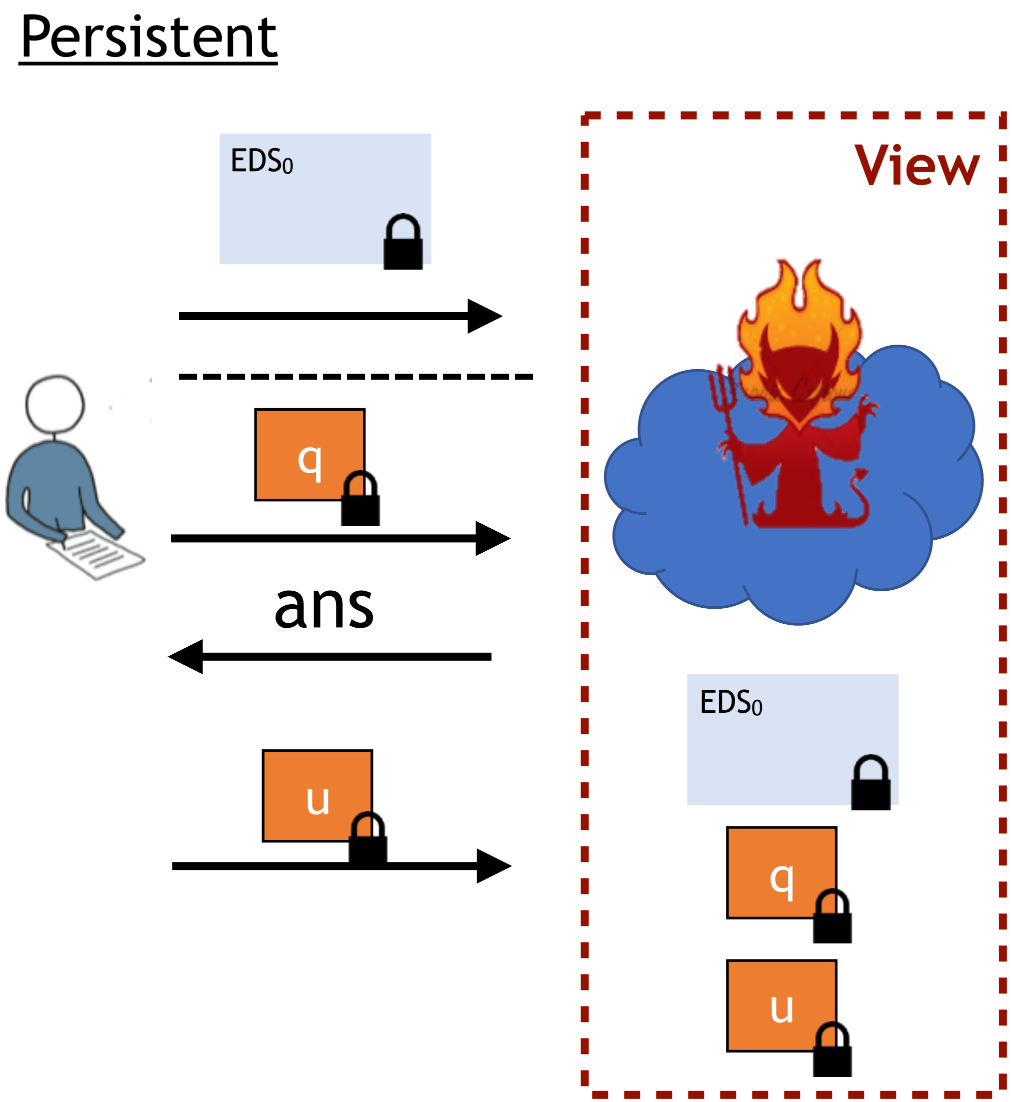
- ‘00 Single-keyword SSE [SWP00,Goh03,CGKO06,CJJJKRS14]
- ‘06 Multi-user SSE [CGKO06,JJKRS13,PPY16,HSWW18]
- ‘13 Boolean SSE [CJJKRS13,PKVK+14,KM17]
- ‘14 Range SSE [PKVK+14,FJKNRS15]
- ‘18 STE-based SQL [KM18]

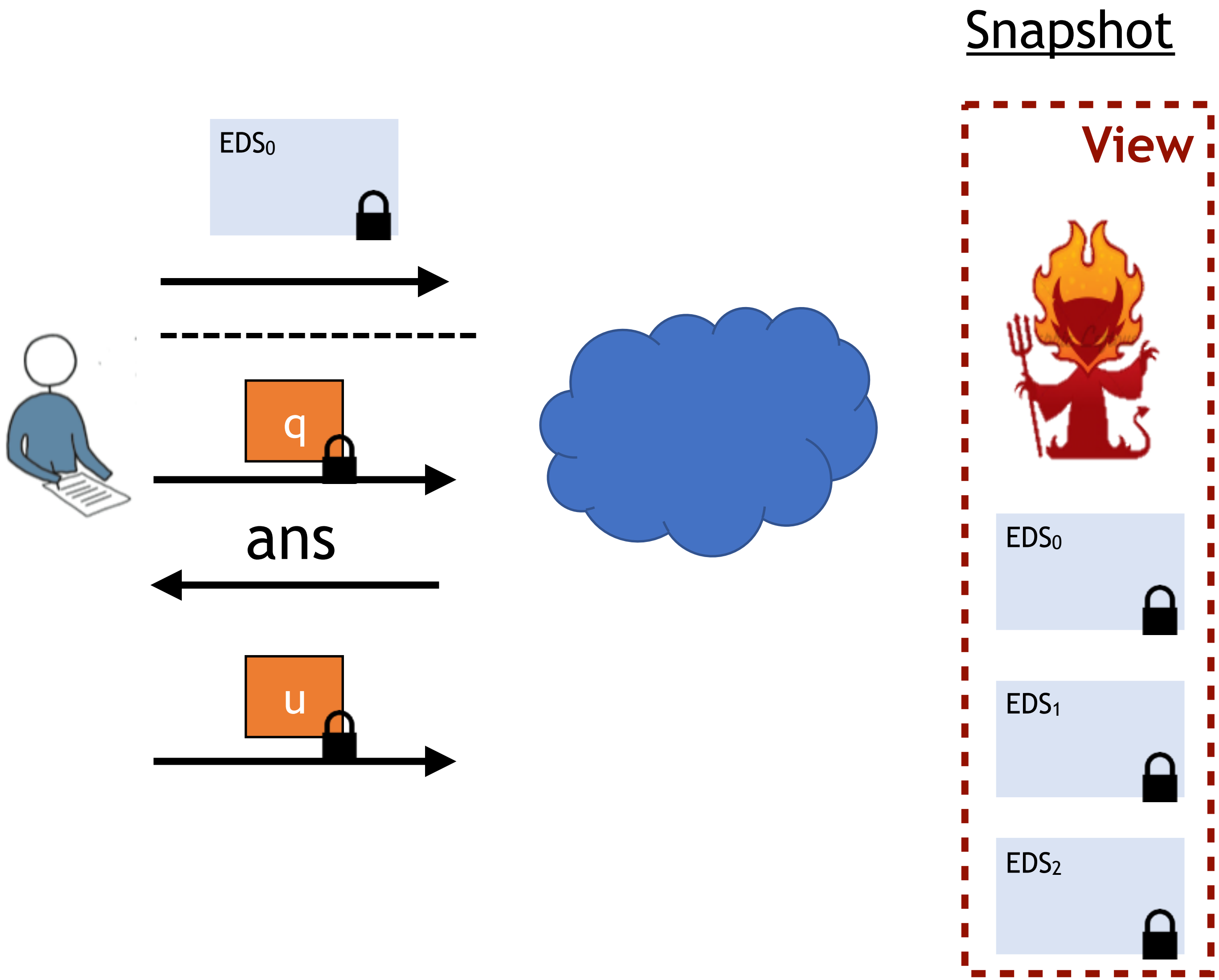
Security

- ‘06 Leakage-parametrized security definitions [CGKO06]
- ‘12 Attacks [IKK12,CGPR15,ZKP16,KMNO16,LMP18,GLMP18]
- ‘14 Forward/Backward Security [SPS14,Bost16,LC17,BMO17,AKM18]
- ‘18 Leakage Supression [KMO18,KM19]
- ‘19 Snapshot [AKM18]

Adversarial Models

Adversarial Models





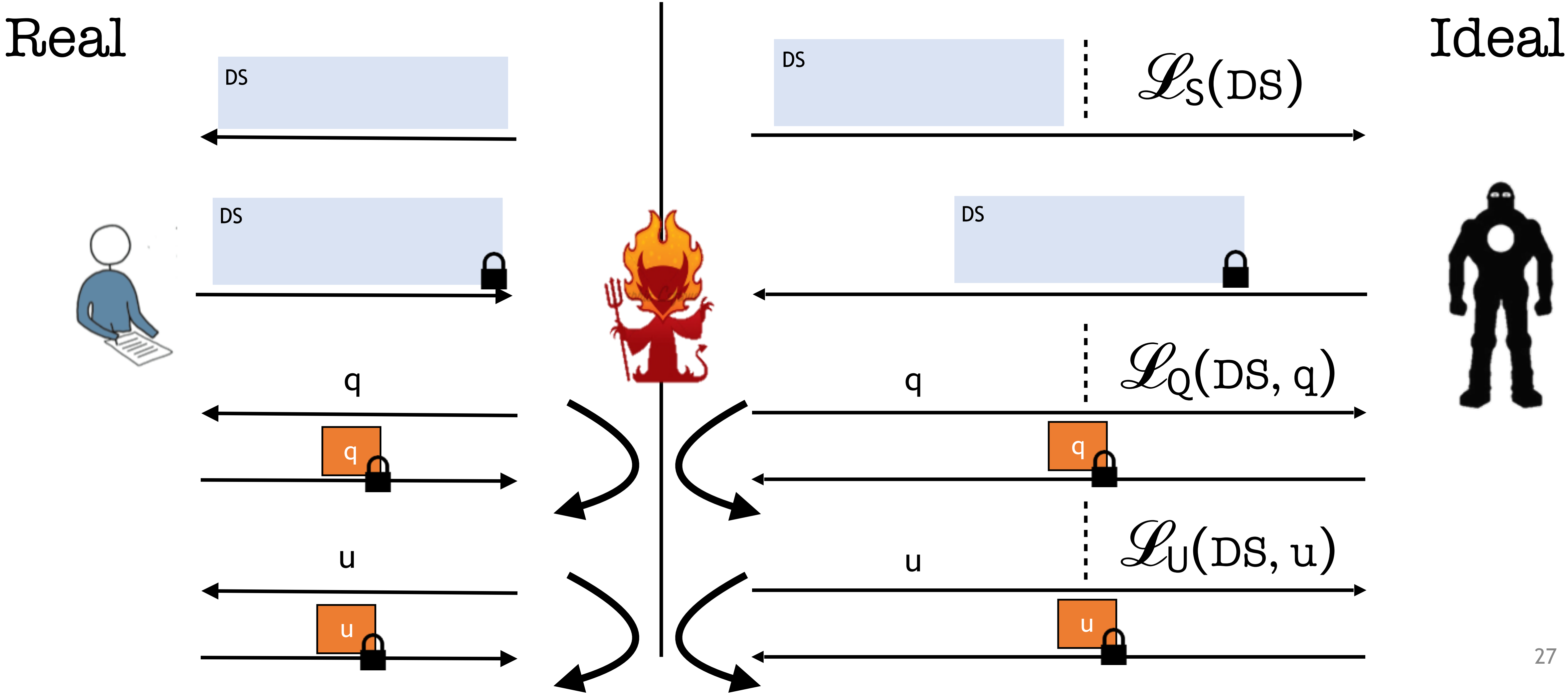
Persistent (Adaptive) Security

[Curtmola-Garay-K.-Ostrovsky06,Chase-K.10]

- An STE scheme is ($\mathscr{L}_S$, $\mathscr{L}_Q$)-secure vs. a persistent adv. if
  - it reveals no information about the *structure* beyond $\mathscr{L}_S$
  - it reveals no information about the *structure* and *query* beyond $\mathscr{L}_Q$

Persistent (Adaptive) Security

[Curtmola-Garay-K.-Ostrovsky06,Chase-K.10]



Forward Privacy

[Stefanov-Papamanthou-Shi14, Bost16]

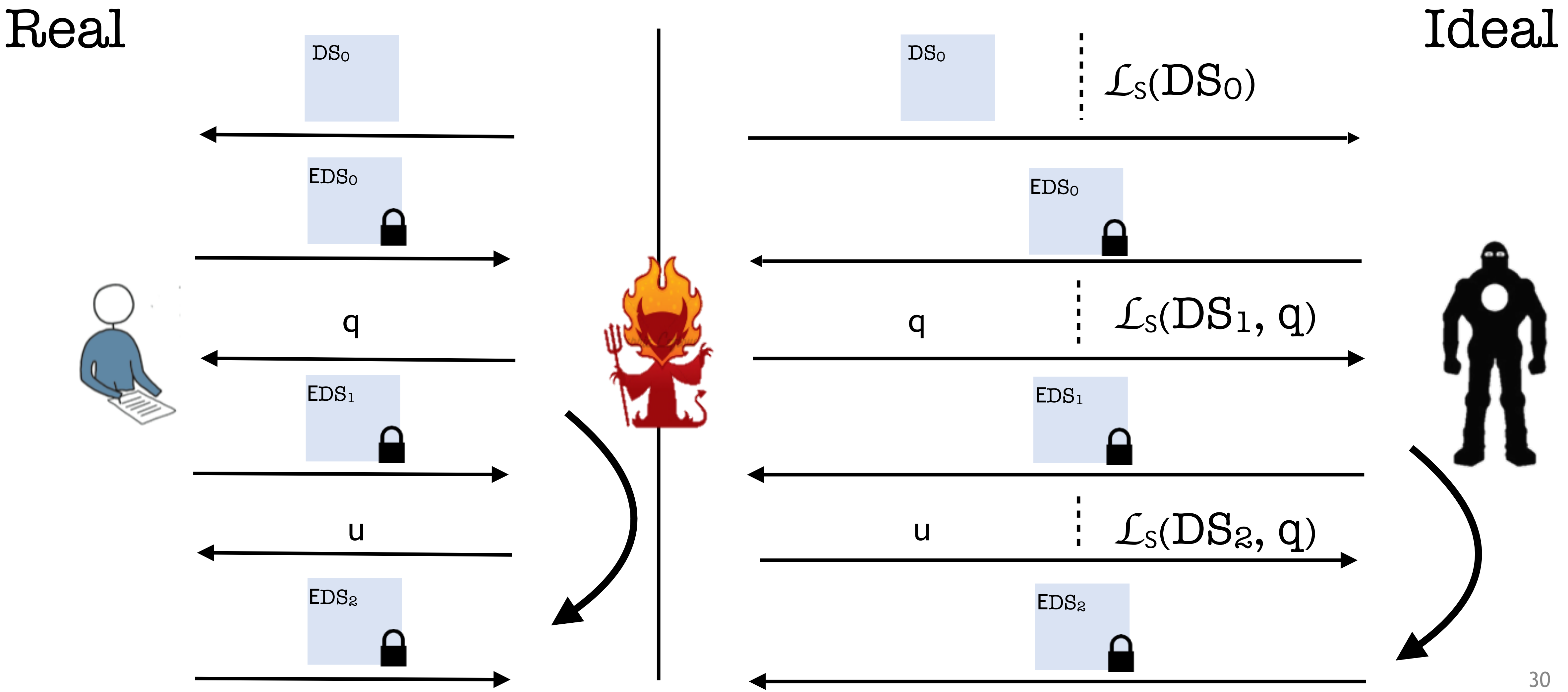
- Informally [SPS14]
  - "Updates not correlated to previous queries"
- Formally [Bost16]
  - $\mathscr{L}_{\mathsf{U}}(\mathsf{MM}, (\ell, \mathbf{v})) = \#\mathbf{v}$

Snapshot (Adaptive) Security

[Amjad-K.-Moataz19]

- We say that an STE scheme is $\mathscr{L}_{\text{Snp}}$-secure vs. a snapshot adv. if
  - it reveals no information about the *structure* beyond $\mathscr{L}_{\text{Snp}}$

Snapshot (Adaptive) Security
[Amjad-K.-Moataz19]



Snapshot (Adaptive) Security

[Amjad-K.-Moataz19]

Static Structures

$$\mathscr{L}_{\mathsf{Snp}} = \mathscr{L}_{\mathsf{S}}$$

Dynamic Structures

Forward privacy $\not\Rightarrow$ Snapshot security

Snapshot security $\Rightarrow$ Insertion independence (variant of history independence)

Snapshot security $\Rightarrow$ Write-only obliviousness

Q: Why do we parameterize definitions with leakage?

Leakage-Parameterized Definitions

[Curtmola-Garay-K.-Ostrovsky, Chase-K.10]

- This area is about tradeoffs
  - but traditional cryptographic definitions don't capture tradeoffs
- in 00's, different approaches were proposed to capture leakage
  - #1: limit adversary's power in the proof
  - #2: make assumptions on data (e.g., high entropy)
- Original motivations for leakage-parameterized definitions
  - Approaches #1 & #2 are misleading (sweep leakage under the rug)
  - Leakage should be made explicit and not be implicit
    - gives clear target for cryptanalysis
    - makes it (somewhat) easier to compare schemes

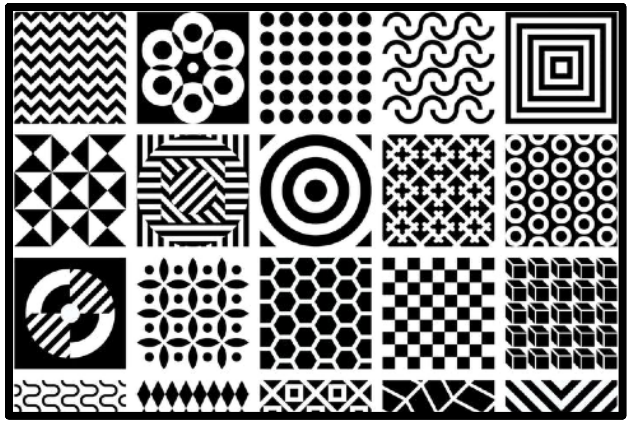
Q: How do we model leakage?

Modeling Leakage

- Each scheme has a leakage profile: $\mathbf{\Lambda} = (\mathscr{L}_S, \mathscr{L}_Q, \mathscr{L}_U)$
  - where $\mathscr{L}_S = (\text{patt}_1, \ldots, \text{patt}_n)$ is the Setup leakage
  - $\mathscr{L}_Q = (\text{patt}_1, \ldots, \text{patt}_n)$ is the Query leakage
  - $\mathscr{L}_U = (\text{patt}_1, \ldots, \text{patt}_n)$ is the Update leakage
- Each “operational” leakage is composed of leakage patterns
  - $(\text{patt}_1, \ldots, \text{patt}_n)$

Common Leakage Patterns

[K.-Moataz-Ohrimenko18]



- **qeq**: query equality
  - a.k.a. search pattern
- **rid**: response identity
  - a.k.a. access pattern
- **qlen**: query length
- **trlen**: total resp. length
- **rlen/vol:** response length
  - a.k.a. volume pattern
- **req**: response equality
- **mqlen**: max query length
- **mrlen**: max resp. length
- **srlen**: sequence resp. length
- **dsize**: data size
- **usize**: update size
- **did**: data identity

Example Leakage Profiles

- The “Baseline” leakage profile for response-revealing EMMs
  - $\mathbf{\Lambda} = (\mathscr{L}_{\mathrm{S}}, \mathscr{L}_{\mathrm{Q}}, \mathscr{L}_{\mathrm{U}})$ = (dsize, (qeq, rid), usize)
- The “Baseline” leakage profile for response-hiding EMMs
  - $\mathbf{\Lambda} = (\mathscr{L}_{\mathrm{S}}, \mathscr{L}_{\mathrm{Q}}, \mathscr{L}_{\mathrm{U}})$ = (dsize, qeq, usize)
- Several new constructions have better leakage profiles
  - AZL and FZL [K.-Moataz-Ohrimenko18]
  - VLH and AVLH [K.-Moataz19]

Structured Encryption vs. Other Primitives

- Encrypted structures appear implicitly throughout crypto
- Oblivious RAM can be viewed as a
  - response-hiding encrypted array
  - with leakage profile $\mathbf{\Lambda}_{\text{ORAM}} = (\mathcal{L}_S, \mathcal{L}_Q, \mathcal{L}_U) = (\text{dsize}, \perp)$
- PIR can be viewed as a
  - response-hiding encrypted array
  - with leakage profile $\mathbf{\Lambda}_{\text{PIR}} = (\mathcal{L}_S, \mathcal{L}_Q, \mathcal{L}_U) = (\text{did}, \perp)$
- Garbled gates can be viewed as
  - response-revealing 2x2 arrays
  - $\mathbf{\Lambda}_{\text{GG}} = (\mathcal{L}_S, \mathcal{L}_Q, \mathcal{L}_U) = (\text{dsize}, \text{qeq})$

Encrypted Multi-Maps

Encrypted Multi-Maps:

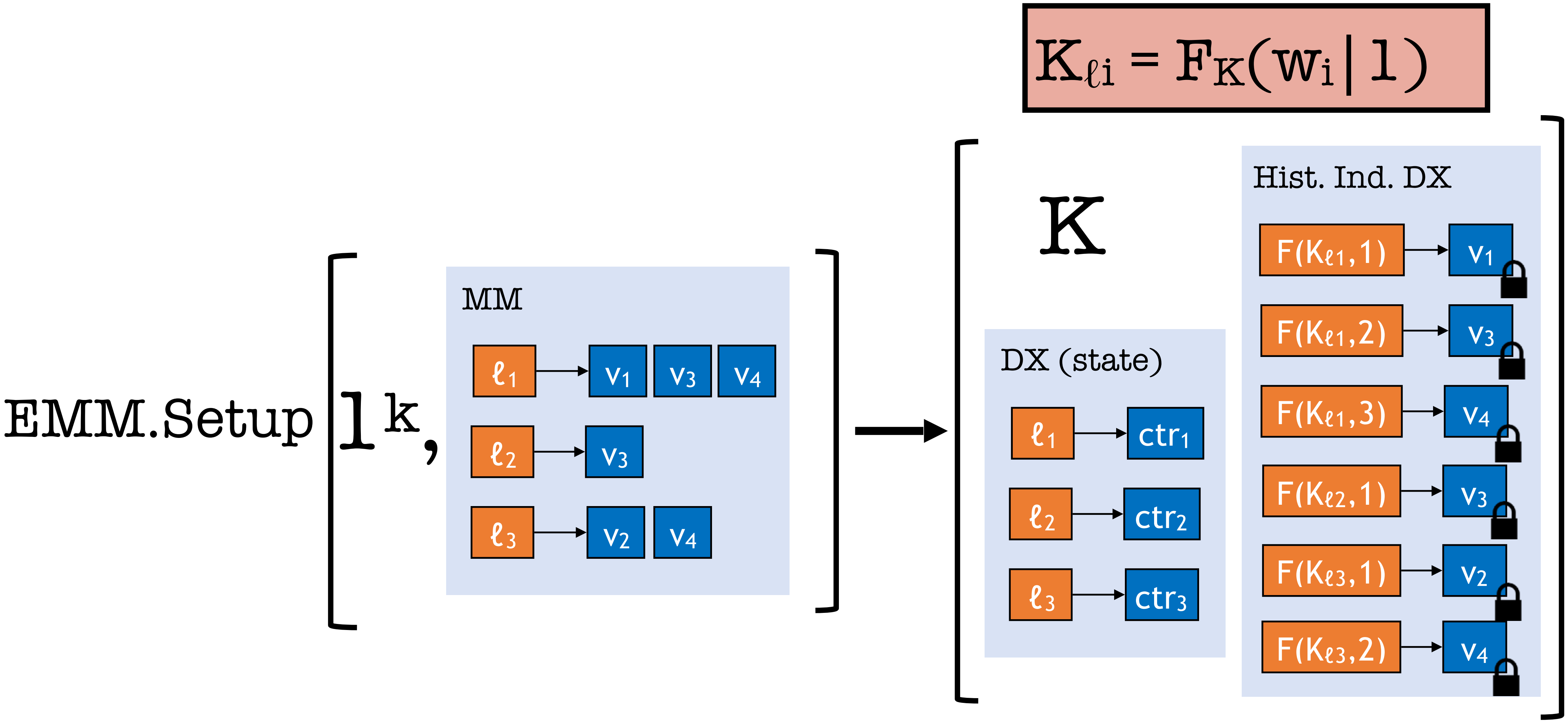
The Heart of *Sub-Linear* Encrypted Search

- EMMs are used as building block for sub-linear
  - Single keyword search [Curtmola-Garay-K.-Ostrovsky06,…]
  - Conjunctive keyword search [Cash et al.13,…]
  - Boolean keyword search [Cash et al.13, K.-Moataz17,…]
  - Range queries [Faber et al.14, Demertzis et al. 16,…]
  - Substring, wildcard, [Faber et al.14,…]
  - SQL databases [K.-Moataz18,…]
  - Graph databases [Chase-K.10,…]

Pidyn (Modified)

[Cash et al.14]

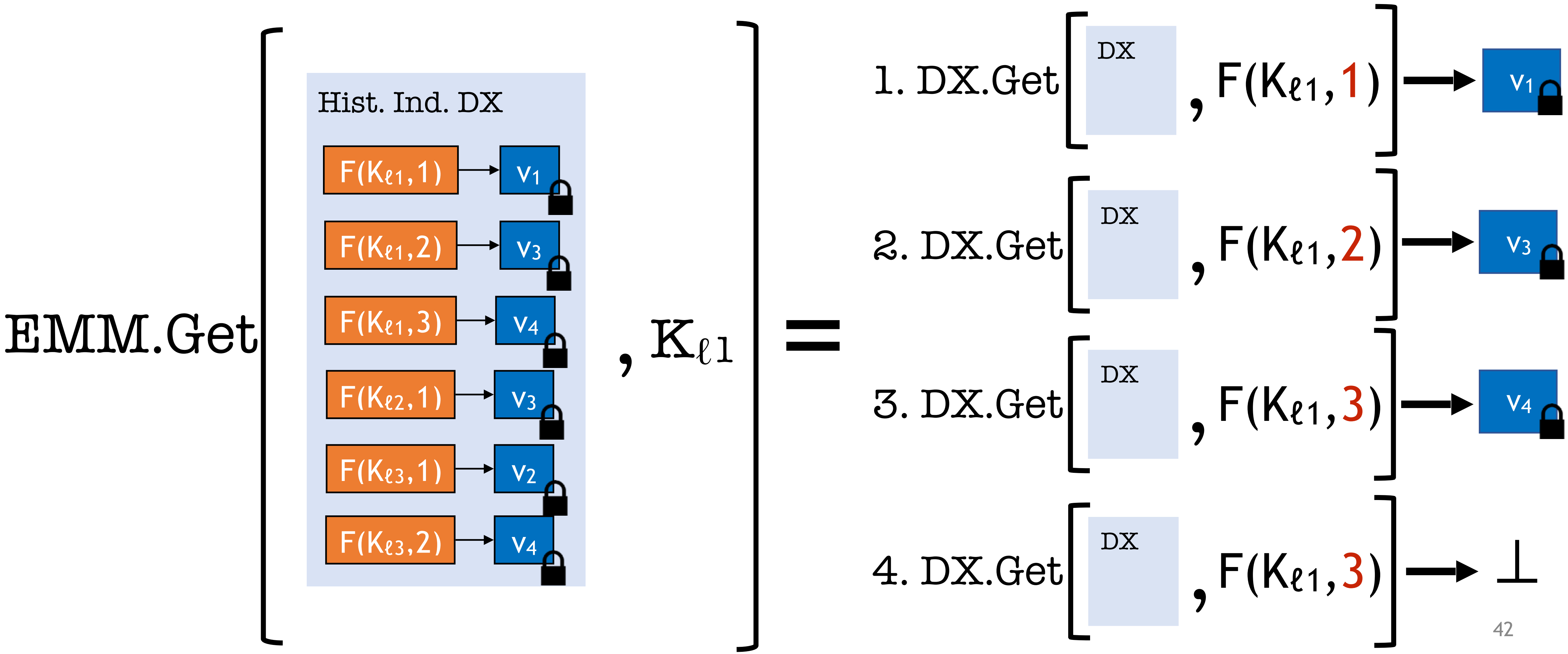
Setup



Pidyn (Modified)

[Cash et al.14]

Get

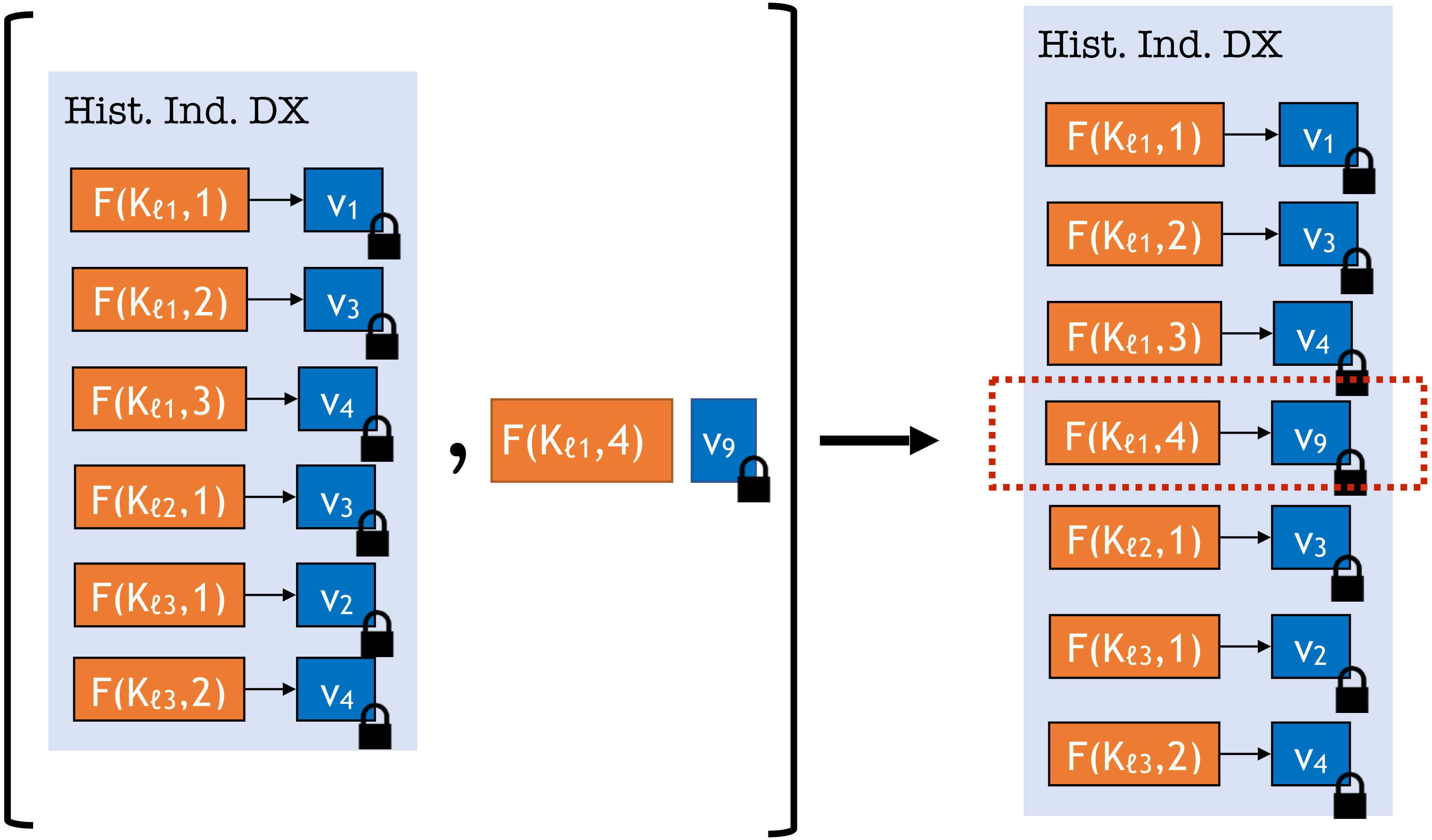


Pidyn (Modified)

[Cash et al.14]

Edit+

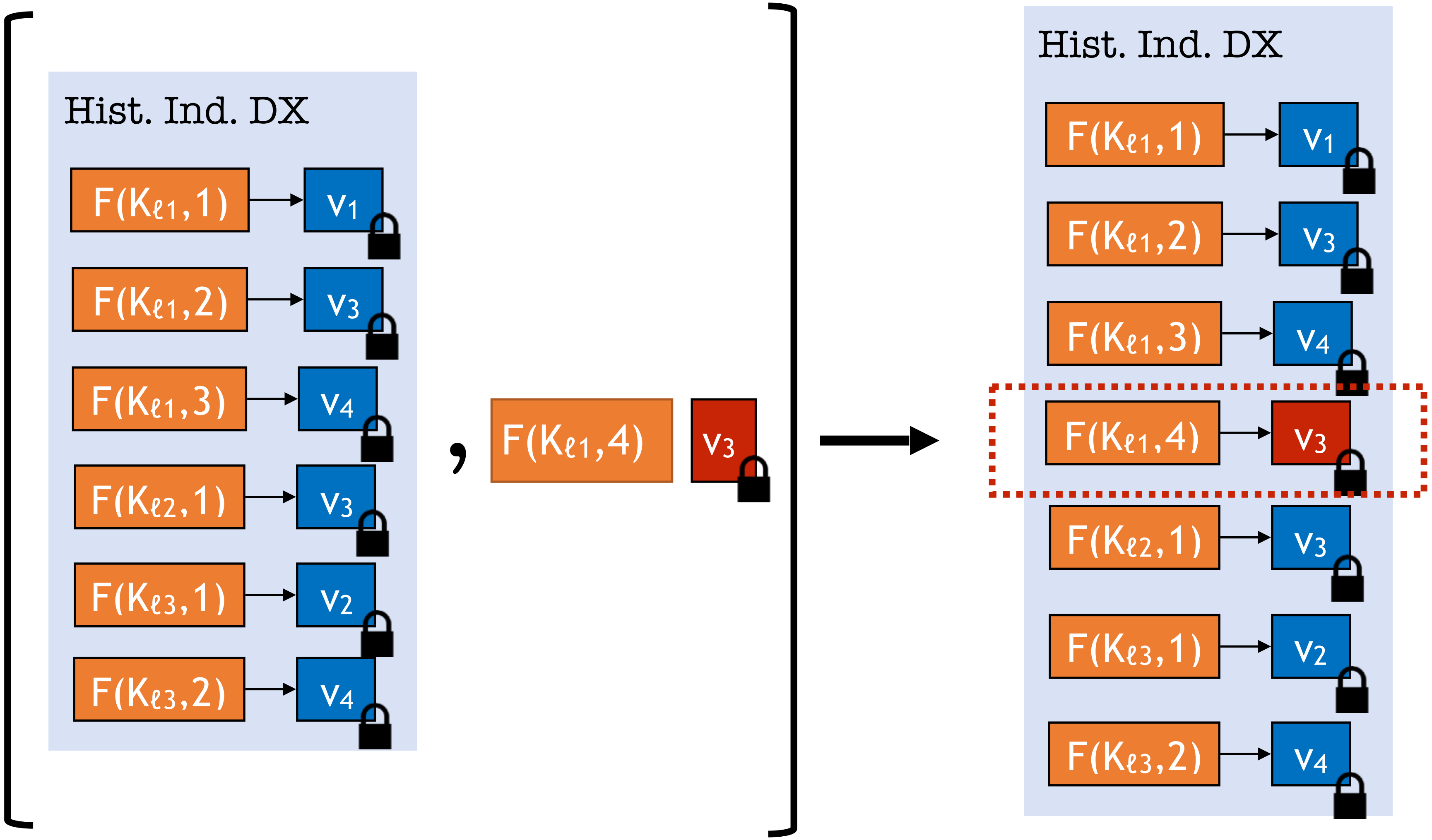
EMM.Edit+



Pidyn (Modified)
[Cash et al.14]

Edit-

EMM.Edit-



Pidyn (Modified)

[Cash et al.14]

Get

K

DX (state)

- $\ell_1 \rightarrow ctr_1$
- $\ell_2 \rightarrow ctr_2$
- $\ell_3 \rightarrow ctr_3$

$\ell_1 = F_K(\ell_1 | 1) = K_{\ell 1}$

v_1 v_3 v_4 v_3

Query complexity:

$O(\#MM[\ell] + dels_0(\ell))$

Storage complexity:

$O(\sum_\ell \#MM[\ell] + dels_0(\ell))$

Hist. Ind. DX

- $F(K_{\ell 1},1) \rightarrow v_1$
- $F(K_{\ell 1},2) \rightarrow v_3$
- $F(K_{\ell 1},3) \rightarrow v_4$
- $F(K_{\ell 1},4) \rightarrow v_3$
- $F(K_{\ell 2},1) \rightarrow v_3$
- $F(K_{\ell 3},1) \rightarrow v_2$
- $F(K_{\ell 3},2) \rightarrow v_4$

I/O Efficiency & Locality

[Cash et al.14]

- The problem with large data
  - if data is very large it gets stored on disk
- Disk seeks are very slow
  - minimize *locality*: # of non-contiguous accesses
  - minimize read efficiency: how much *additional* data is read
    - reading *contiguous* data is OK but not too long
- Pidyn has poor locality
  - **Get(ℓ)** needs **#MM[ℓ]** non-contiguous accesses

I/O Efficiency & Locality

[Cash et al.14]

- Introduce several schemes with improved locality
  - **Pipack**: packs values in a single ciphertext
  - **Piptr**: packs pointers to values in a single ciphertext
    - this tradeoffs EMM locality for standard memory locality
  - **2Lev**: combines both techniques

Local SSE Schemes

- [Cash-Tessaro14]
  - lower bounds for “non-overlapping” schemes (improved by Asharov et al.)
- [Asharov-Segev-Shahaf18]
  - lower bound for “pad-and-split” schemes
  - **$L(N)$** locality & **$O(1)$** read efficiency $\Longrightarrow$ **$\Omega(N \log N / \log L)$** space
  - matched by [Demertzis-Papamanthou17]
- [Asharov-Naor-Segev-Shahaf18]
  - lower bound for “statistically-ind.” schemes
  - **$O(1)$** locality & **$O(N)$** space $\Longrightarrow$ **$\omega(1)\cdot\varepsilon(n)^{-1}$** read efficiency
  - matched by [Asharov-Segev-Shahaf18]

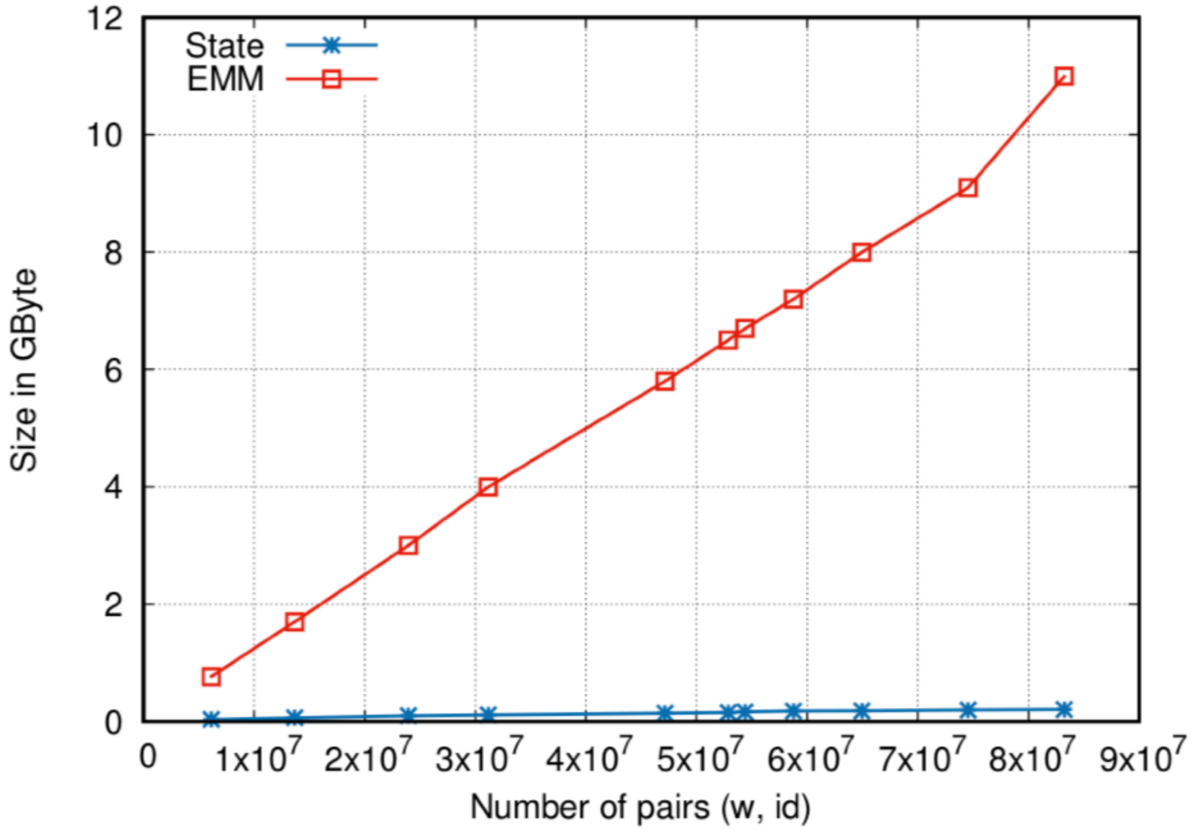
Limitations of Pidyn, Pipack, Piptr, 2Lev

- Not forward private
  - update tokens can be linked to previous search tokens
  - can be exploited using adaptive file injection attacks
- Query and storage complexity depend on total # of deletes

State-of-the-Art EMMs

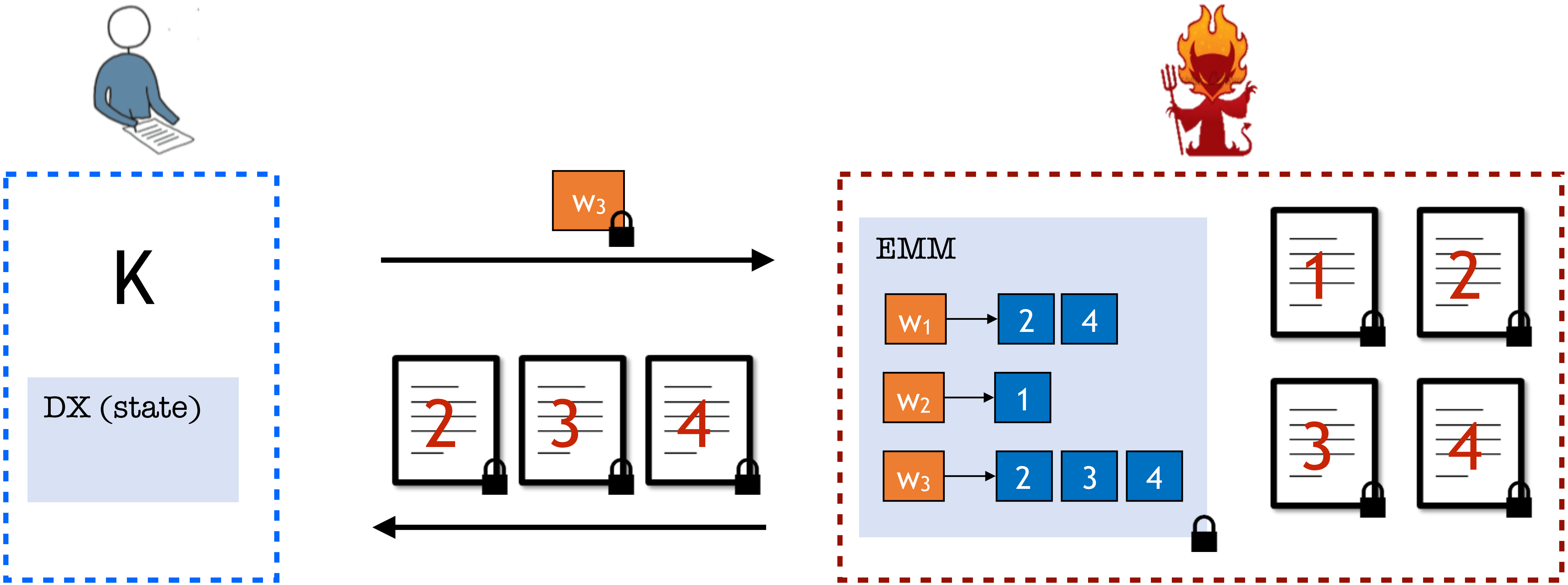
	Search	Client Storage	Forward Privacy	Snapshot
SPS'14	O(#MM[ℓ]·polylog(#MM[ℓ]))	O(#𝕃)	Yes	Yes
B'16	O(#MM[ℓ] + dels_0(w))	O(#𝕃)	Yes	No
BMO'17	O(#MM[ℓ] + dels_0(w))	O(#𝕃)	Yes	No
EKPE'17	O(#MM[ℓ] + dels_s(w))	O(#𝕃)	Yes for adds No for dels	No
AKM19	O(#MM[ℓ] + dels_r(w))	O(#𝕃 + ML)	Yes	Yes

[AKM19] Client State



- EDB w/ 83 million pairs (11GB)
  - state is 210MB

Single Keyword Search from EMMs



Sub-Linear Constructions from Black-Box EMMs

- Searchable symmetric encryption [Curtmola-Garay-K.-Ostrovsky06,…]
- Graph queries [Chase-K.10,…]
- Conjunctive & disjunctive keyword search [Cash et al. 13,]
- Worst-case sub-linear disjunctive & Boolean search [Pappas et al.14, K.-Moataz17]
- Wildcard & substring search [Faber et al.15]
- Range search [Faber et al.15,Demertzis et al.16,Podar-Boelter-Popa16]
- SQL queries on relational DBs [K.-Moataz18]

Sub-Linear Constructions from Black-Box EMMs

- Why constructions based on black-box EMMs?
- Modularity
  - easy to design, understand and analyze
  - benefit from improvements in EMM efficiency
  - benefit from improvements in EMM security/leakage