Searchable Encryption

Hugo Krawczyk

IBM Research

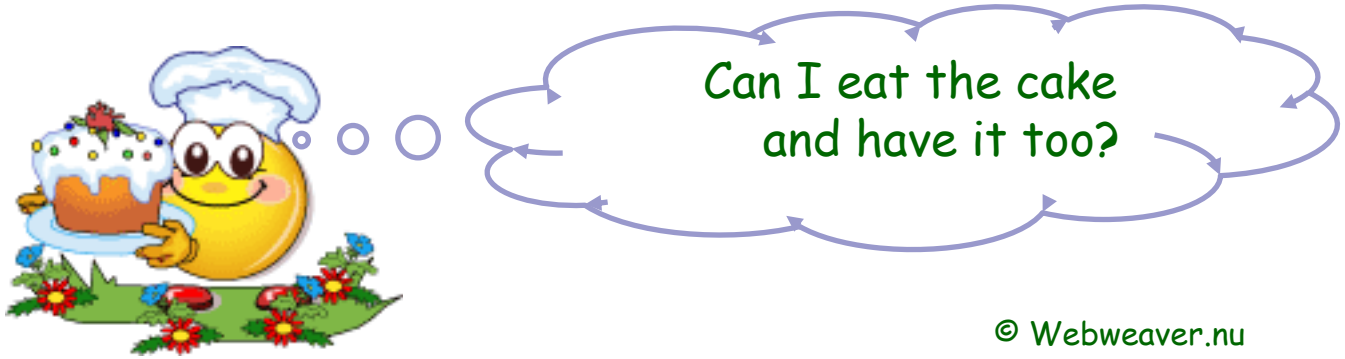
Winter School – Bar Ilan University – January 2015

Class Plan (time permitting)

- Part 1: Overview/Intro (BIU-hugo-1-overview.pdf)
  - The searchable encryption problem, models and functionalities
  - Dedicated solutions and state of the art (OXT Protocol)
- Part 2: The OXT single-client protocol
  - Single keyword search (BIU-hugo-2-SKS.pdf)
  - Conjunctions and Boolean queries (BIU-hugo-3-OXT.pdf)
  - Range and substring queries (BIU-hugo-4-complex.pdf)
- Part 3: Multi-client and OSPIR settings (BIU-hugo-5-OSPIR.pdf)
- Part 4: Other solutions, attacks and research questions
  (Slides at the end of BIU-hugo-1-overview.pdf)

The Data-in-the-Cloud Conundrum

- Your data in the cloud: email, backups, financial/medical info, etc.
- Data is visible to the cloud and to anyone with access (legitimate or not)
  - At best, data is encrypted "at rest" with the server's keys and decrypted upon use
- Q: Why not encrypt it with your (data owner) own keys?
- A: Utility, e.g. allow the cloud to search the data (e.g. gmail)
- **Can we keep the data encrypted and search it too?**

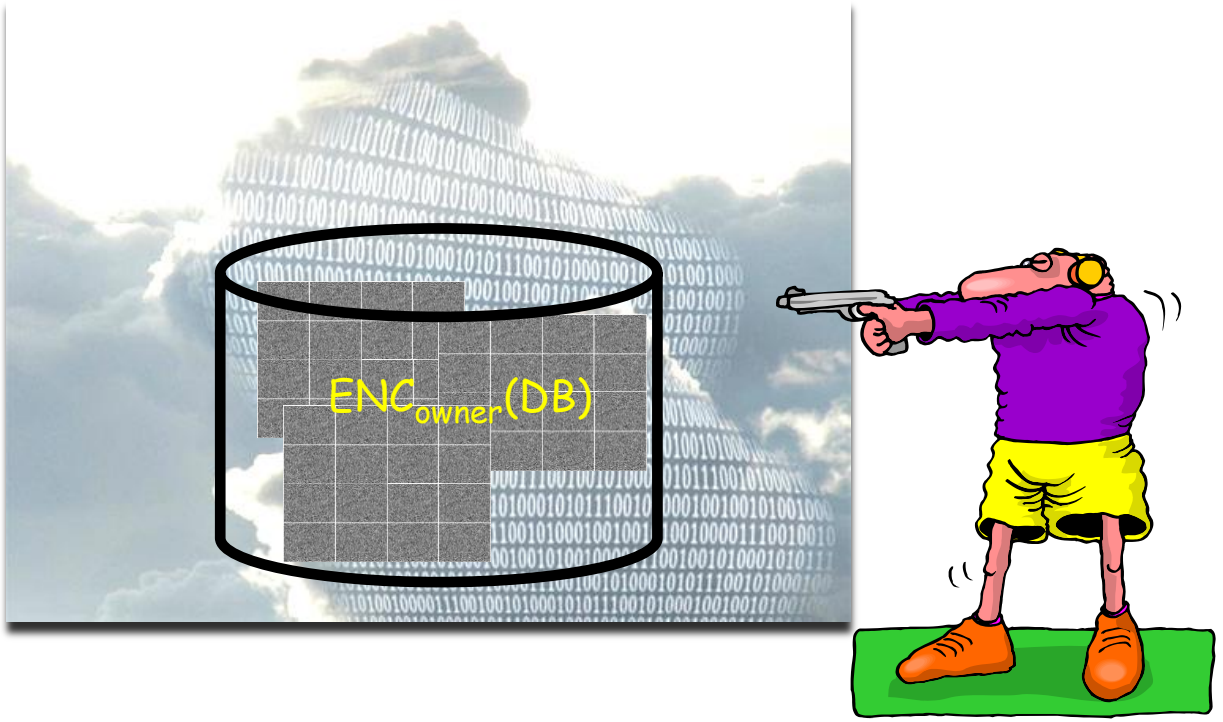


SSE: Searchable Symmetric Encryption

- Owner outsources data to the cloud: Pre-processes data, stores the processed and encrypted data at the cloud server
  - Keeps a small state (e.g. a cryptographic key)
  - Later, sends encrypted queries to be searched by the server
    - e.g. return all emails with *Alice as Recipient, not sent by Bob, and containing at least two of the words {searchable, symmetric, encryption}*
- Goal: Server returns the encrypted matching documents w/o learning the plaintext query or plaintext data
  - Some forms of statistical leakage allowed: data access patterns (e.g. repeated retrieval, size info), query patterns (e.g., repeated queries), etc.
    - Plaintext data/queries *never directly exposed*, but *statistical inference* possible
- Protects against break-ins, cloud insiders, even "surveillance attacks"

With SSE…

The cloud cannot disclose your data… ***not even at gun point!***



SSE before 2013

- Generic tools: FHE, ORAM, PIR
  - Expensive
    - BUT ORAM getting closer to practice for moderate size DBs (Benny Pinkas talk)
  - great* security
    - *assumes *all raw data* is ORAM-encrypted, o/w leakage via access patterns
- Deterministic + order preserving encryption: e.g. CryptDB [PRZB'11]
  - Practical but significant leakage (Naveed-Kamara-Wright, CCS'2015)

Deterministic and order preserving

Name	Lastname	Age
Elaine	Samuels	24
Mary	Stein	37
Jim	Stein	81
John	Sommers	3
Mary	Williams	17
John	Garcia	43
John	Gould	37

Name	Lastname	Age
Ge5$#u	Q*6sh#	223
E89(%y	2@#3Br	340
2Tr^#7	2@#3Br	736
qM@9*h	gYv6%t	34
E89(%y	X%3oL7	160
qM@9*h	wnM7#1	308
qM@9*h	8vy8$Z	340

Attack on CryptDB

- Methodology

"Inference Attacks against Property-Preserving Encrypted Databases" Naveed-Kamara-Wright. *CCS'2015.*

  - Input: CryptDB-encrypted medical database DB_1 (hundreds of hospitals)
  - Training data: Plaintext medical data from public database DB_2
  - Output: Decrypted DB_1 data via correlation analysis using DB_2 as training data

- Basic attacks: frequency, sorting and cumulative analysis
- Results (for each analyzed column): "at least x% correctly decrypted records in y% of the hospitals" (denoted x/y). Examples:
  - Race=60/69 (race guessed correctly for at least 60% of patients in 69% of the hospitals – race admits 6 values)
  - Major Diagnostic Category = 40/27 (admits 25 values)
  - Age 95/78 (125 possible values)

SSE before 2013

- Generic tools: FHE, ORAM, PIR
  - Expensive (ORAM getting closer to practice for moderate size DBs)
  - great* security
    - *assumes *all* raw data is ORAM-encrypted, o/w leakage via access patterns
- Deterministic + order preserving encryption: e.g. CryptDB [PRZB'11]
  - Practical but significant leakage (Naveed-Kamara-Wright, CCS'2015)
- Name of the game: Security-Functionality-Performance

Tradeoffs

SSE before 2013 (cont.)

- Dedicated SSE solutions*:
  - Single-Keyword Search (SKS) [SWP'00, Goh'03, CGKO'06, ChaKam'10, …]
    - "privacy optimal" (if we don't count encrypted query results as leakage)
  - Conjunctions: Very little work
    - naive (n single-keyword searches),
    - GSW'04: structured-data, LINEAR in DB, communication-pairings tradeoff
- Practicality limitations
  - single-keyword only support, limited support for dynamic data
  - non-scalable design (esp. adaptive solutions), no I/O support for large DBs
  - little experimentation/prototyping

* Survey: Bosch-Hartel-Jonker-Peter ACM Comput. Surv. 47, 2, Article 18
http://eprints.eemcs.utwente.nl/24788/01/a18-bosch.pdf

ESPADA/OXT (our technical focus)

- Joint work IBM-UCI teams:
- David Cash, Sky Faber, Joseph Jaeger, Stas Jarecki, Charanjit Jutla, Quan Nguyen, Marcel Rosu, Michael Steiner
- Crypto'13, CCS'13, NDSS'14, ESORICS'15
- IARPA SPAR Program
  - Reduce agencies' reluctance to share information (9/11, Boston bombing)
  - Preparing for a post-PATRIOT world (DHS has a "chief privacy officer") *

Snowden

* alturl.com/ot72x

- Co-performers: Columbia + Bell Labs - Blind Seer [Oakland 14 + 15]
  - Guest presentation: Ben Fisch

ESPADA: Extends SSE in 4 dimensions

1. Functionality (well beyond single-keyword search):

   - Conjunctions
   - General Boolean expressions (on keywords)
   - Range queries
   - Substring/wildcard queries, phrase queries

   Search on *structured data* (relational DBs) as well as *free text*

2. Scalability:

   - *terabyte*-scale DB, *millions* documents/records, *billions* indexed document-keyword pairs
   - *Dynamic* data
   - Validated implementation, tested by a third party (IARPA, Lincoln Labs)

3. Provability: "imperfect security" but with provable leakage profiles (establishing upper bounds on leakage), well-defined adversarial models

This work: extends SSE in 4 dimensions

4. New application settings and trust models

- Multiple clients: Data owner D outsources Encrypted DB to cloud; clients run queries at the cloud but *only for queries authorized* by D
  - Leakage to cloud as in basic SSE, client only learns documents matching authorized queries (policy-based authorization enforced by data owner)
- Blind authorization: As above but authorizer enforces policy without learning the queried values (we call it "*Outsourced Symmetric PIR*")
  - Assumes non-collusion between cloud and data owner
- Note: multi-reader, single-writer system (no public key encryption)

Example Applications

- Example: Hospital outsources DB, provides access to clients (doctors, administrators, insurance companies, etc.)
  - Policy-based authorization on a client/query-basis
  - Hospital doesn't need to learn the query, only (blindly) enforce policy
    - Good for security, privacy, regulations
- Warrant scenario (extended 4-party setting)

Obama's 3rd Party Solution (phone data)

  - Judge provides warrant for a client C (e.g. FBI) to query a DB
  - DB owner enables access but only to queries allowed by judge
  - DB owner does not learn warrant content or queries
  - Client C (e.g., FBI) gets the matching documents for the allowed queries and nothing else

Large-Scale & Functional Implementation (OXT)

- Support for arbitrary Boolean queries on all 3 (extended) SSE models
- Validated on synthetic census data: 10Terabytes, 100 million records, > 100,000,000,000=10^{11} indexed record-keyword pairs !
  - Equivalent to a DB with one record for each American household and 1000 keywords in each record and any boolean query (including textual fields)
  - Smaller DB's: Enron email repository, ClueWeb (>> English Wikipedia)
- Support for range queries, substring/wildcards, phrase queries (5x perf. cost)
- Dynamic data: Supports additions, deletions and modifications of records

Scalability

- Preprocessing scales linearly w/ DB size (minutes-days for above DBs)
  - Careful data structure, crypto and I/O optimizations
    - Can benefit on any improvement on single-keyword search
- Search proportional to # documents matching the least frequent term: $w_1 \wedge B(w_2, \ldots, w_n)$ (w_1 called the s-term)
  - *Single round* to retrieve matching document indexes (tokens from client to server, matching indices back; retrieve encrypted documents)
  - Query response time: Competitive w/ plaintext queries on indexed DB

4 seconds: fname='CHARLIE' AND sex='Female' AND
NOT (state='NY' OR state='MA' OR state='PA' OR state='NJ')
on 100M records/22Billion index entries US-Census DB

Crypto Design-Engineering Synergy

- Major effort to build I/O-friendly data structures
  - Critical decision: Do not design for RAM-resident data structures (it severely *limits scalability*)
  - Challenge: need to avoid random access (e.g., avoid Bloom filters on disk)
    - Need *randomized* data structures to reduce leakage and need *structured* ones to improve I/O performance (locality of access)
- Cryptographic index based on elliptic curve cryptography (optimized for very fast exponentiation, esp. with same-base)
  Typically: I/O and network latency dominate cost

  500,000/sec, 8 cores, same-base opt , 100-1000 per IO
  - On a midsize storage system: ~300 IOPS (I/O Operations Per Second)
  - ~1000 expon's per random I/O access (133 w/o same-base optimization)
- Data encryption uses regular symmetric crypto (e.g., AES)

Security: The challenge of being imperfect

- Good news: Semantic security for data; no deterministic or order preserving data encryption
- But: Security-Performance trade-offs → Leakage to server
  - Leakage in the form of access patterns to retrieved data and queries
    - Data is encrypted but server can see intersections b/w query results (e.g. identify popular document, intersection b/w results of two ranges, etc.)
    - Server learns query function (not values/attrib's); identifies repeated query
  - Additional specific leakage (more complex functions of DB and query history):
    - E.g. we leak $|Doc(w_1)|$ and in query $w_1 \wedge w_2 \wedge \ldots \wedge w_n$ we leak $|Doc(w_1 \wedge w_i)|$
    - E.g. the server learns if two queries have the same w_1 (other terms are hidden)
- Leads to statistical inference based on side information on data (effect depends on application), masking techniques may help

Security: The challenge of being imperfect

- Security proofs: Formal model and precise provable leakage profile
  - Security modeling and definitions follow simulation paradigm [*CGKO, CK*]
  - Leakage profile: provides upper bounds on what's learned by the attacker
- Syntactic leakage vs "semantic leakage"
  - Need to assess on an application basis and relative to a-priori knowledge
    - For example, formal leakage proven even if attacker can choose data and queries – but in practice semantic leakage will be substantial in this case.
      - E.g. Cash, Grubbs, Perry, Ristenpart, *CCS'2015* (Sasha's talk)
    - Even the "basic leakage" from access to encrypted results (e.g., sizes and intersections) can be very significant in some cases
- Yet, we expect in many cases to provide meaningful (if imperfect) security (in particular, relative to property-preserving solutions)

Columbia/Bell Labs Solution (Blind Seer)

- Parallel work: Same IARPA project – papers at [Oakland'14, 15]
- Elegant solution based on Bloom filter trees with Garbled Yao for privacy and authorization
  - Conceptually simpler than ours; e.g., no need to choose s-term
  - Symmetric crypto and multi-party computation techniques (Yao) (instead of homomorphic operations in our case) → much faster pre-processing
  - Less scalable: Bloom filters are *inherently* random access
    →DB sizes limited by the size of RAM
  - Single client, limited negations
  - Many trade-offs with ESPADA (but incomparable leakage)
    - e.g., Bloom filter path vs. w_1-related leakage

Practice

- Is CryptDB (and other DetEnc/OPE solutions) sufficient in practice?
  - Is their leakage acceptable?
- Who is the attacker?
- What do regulations say?
- Is it enough to not being the weakest link?

Practice

- CryptDB (and other DetEnc/OPE solutions) are *legacy friendly* .
  But is their leakage acceptable?
  - Who is the attacker?
  - What do regulations say?
  - Is it enough to not being the weakest link?

Need more "privanalysis" – current attacks just scratch the surface

Closing Remarks and Future Work

(following slides will be presented in the closing class)

Challenges

- Leakage: how do we characterize, prove, evaluate
- Tradeoffs: interplay security-performance (asymptotics & concrete)
  - space/computation/privacy
- Close engineering-theory interaction: keep it simple!
  - can't throw the heavy weapons on the problem
- Prove! Crypto design w/o proof not worth much (especially if you are going to build/use the system)
  - Complicating a proof is fine, complicating a solution is not

Lot of…

- Room for improvement (functionality, privacy, performance)
- Interesting research questions
- Trade-offs to resolve
- Fundamental bounds to be proven
- Theories and models to be developed
- *Privanalysis* attacks to make you famous (easy to get papers accepted…)
- Dealing with "the challenge of being imperfect"
  - Leaving the "all-but-negligible security guarantees" paradise
  - Is there an acceptable compromise? Should we abandon it to ad-hoc practitioners? Too dirty for our souls?

Research Questions (partial list)

- Leveraging other tools (carefully): MPC, ORAM, homomorphic encryp'n
- Fundamental limits (leakage-computation tradeoffs), e.g.:
  - leakage from returned ciphertexts (ORAM helps but at significant cost)
  - Frequency of w_1 (least frequent term) (reduction from 3SUM)
- "Semantic leakage": Proving formal leakage is nice but how bad is it for a given particular application, what forms of masking can help?
  - Can we have a theory to help us reason about it (cf. differential privacy)?
  - A theory of leakage composition? Guidance for masking techniques
  - Attacks welcome! IKK'12, KNW'15, CGPR'15 *just scratched the surface*
- Characterizing *privacy-friendly* plaintext search algorithms/data str.
- A more complete SQL query set (esp. joins)

Tradeoffs

- Tons of privacy-performance trade-offs examples
  - BXT vs OXT vs PXT (computation/communication), Masking s-terms and Xset (space), Bloom filters (false positives), network-latency, and more
  - Fundamental: ???
- Privacy/performance bounds?
  - E.g., the intrinsic cost of perfect secrecy
  - But how about bounds in terms of necessary leakage
    - E.g. in the case of perfect secrecy even a 1 bit of leakage can be really bad
- Any hope for trade-offs between "polynomially-related" objects?

Example: s-term leakage

- OXT leaks the size of $|DB(w_1)|$ (w_1 is the least frequent conj. term)
- Necessary? "Yes". Can you prove it? No.
- Conjecture: any conjunctions algorithm will leak (via running time) an upper bound on $|DB(w_1)|$, except if
  - Search is padded to $\max_w |DB(w)|$ size (➔ search is linear in $|DB|$)
  - Or: Conjunctions pre-computed (➔ pre-processing is super-linear)
- Why? Consider 2 conjunctions that return the same small # of records, one with 2 infrequent terms, one with 2 very frequent terms
  - "name=David and gender=Female" vs "name=Charanjit and lastname=Jutla"
- We conjecture a lower bd on plaintext search (hence on encrypted)
  - Reduction from 3SUM (based on [P'10])

Join the (multi) Party…

- *An exciting & large space to explore with many many research opportunities!*
- *… and many practical applications*
  - Very timely given cloud migration, explosion of private info, and strong attackers (including surveillance, espionage, mafia, and just hackers…)
- *An opportunity for sophisticated crypto in the real world?*

Summary

- Great progress relative to work on single-keyword single-client SSE
  - Rich queries: General Boolean queries (structured data, free text), Plus: range, substring, wildcards, phrase, proximity
  - Huge DBs: 10 TB, 100M records, 10^{11} indexed keyword-document pairs
    - EDB creation linear in DB size, queries competitive with MySQL
  - Single- and Multi-Client models, policy-based delegation of queries
  - Authorization w/o learning query ("Outsourced Symmetric PIR")
  - Privacy, insider security, surveillance protection, warrant enforcement
- Imperfect security: Leakage from access- and query-patterns, but well defined leakage profiles, and simulation-based adaptive security
- Many challenging theoretical and engineering questions
  - Going for practice? Don't forget simplicity, engineering and… proofs!

Thanks!

- Crypto'2013: Boolean search, single client eprint.iacr.org/2013/169
- CCS'2013: Multi-client, Blind authorization eprint.iacr.org/2013/720
- NDSS'2014: Dynamic data, implementation eprint.iacr.org/2014/853
- ESORICS 2015: Range, Substrings, Wildcards, Phrases 2015/927

Backup

Single Keyword Search (SKS)

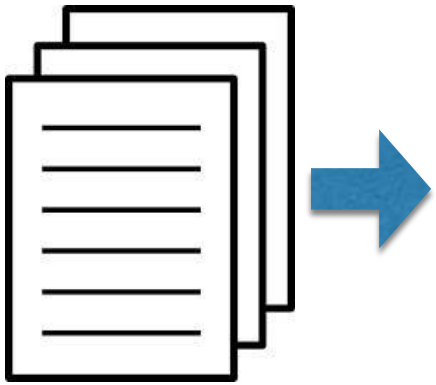
Graphics courtesy of David Cash

Single-Keyword Search (SKS) in SSE

- SSE (searchable symmetric encryption)
  - A client C (both client and data owner) and server E (* Charlie, Eddie *)
  - Client C transforms its plaintext DB into "encrypted DB" (EDB) that includes encrypted records and metadata;
  - EDB stored at server E; C only keeps a cryptographic key
- SKS: Given keyword w return *indices* of documents containing w
  - Important: *model simplified* by abstracting out retrieval (and enc/dec) of documents (note: variable vs. fixed-length ciphertexts)
- SKS at the basis of all our solutions

SKS with Cleartext Lists

1 Encrypted keyword tags



Inverted index

keyword	records
Rutgers	4, 9,37
Admissions	9,37,93,94,95
committee	8,37,89,90
accept	4,37,62,75

Encrypted index

keyword	records
45e8a	4, 9,37
092ff	9,37,93,94,95
f61b5	8,37,89,90
cc562	4,37,62,75

1. Build inverted index (each keyword points to record id's)
2. Choose key K and replace each keyword with PRF tag F(K,w)
3. Client saves key K

2 Search protocol

1. Client sends F(K,w)
2. Server retrieves proper row

SKS with encrypted lists

- additionally encrypt rows under different keys
- requires modification of server, but more secure

45e8a	4, 9,37
092ff	9,37,93,94,95
f61b5	8,37,89,90
cc562	4,37,62,75

45e8a	
092ff	
f61b5	
cc562	

There should no leakage on individual size of record lists

keyword	records
45e8a	
092ff	
f61b5	
cc562	
a845c	
b8423	
ab067	
63fa2	
54db1	
b7696	
ed15b	

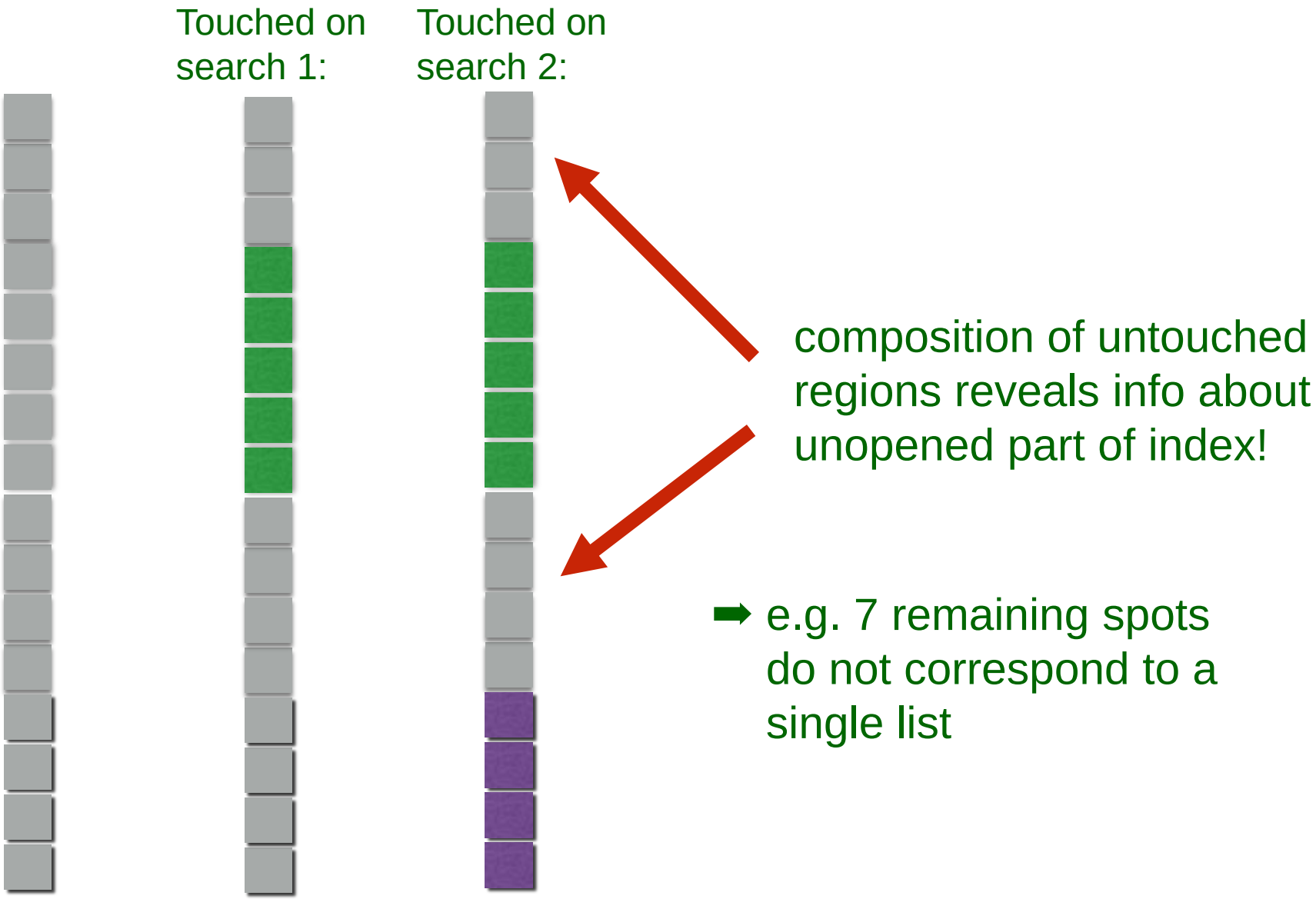
nCeUKlK7GO5ew6mwpIra
ODusbskYvBj9GX0F0bNv
puxtwXKuEdbHVuYAd4mE
ULgyJmzHV03ar8RDpUE1
6TfEqihoa8WzcEol8U8b
Q1BzLK368qufbMMHlGvN
sOVqt2xtfZhDUpDig8l0
jyWyuOedYOvYq6XPqZc2
5tDHNCLv2DFJdcD9o4FD

How to maps lists into memory

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

keyword	records
K_1	
K_2	
K_3	
K_4	

server can observe memory touched during searches:



A distinguishing example

Index I_0

Rutgers	4,9,37
Admissions	9,19,93,94,95
Committee	8,76,89,90
Accept	2,35,62,75

1. Search $w_{0,1}$ = “Rutgers”
2. Search $w_{0,2}$ = “Admissions”

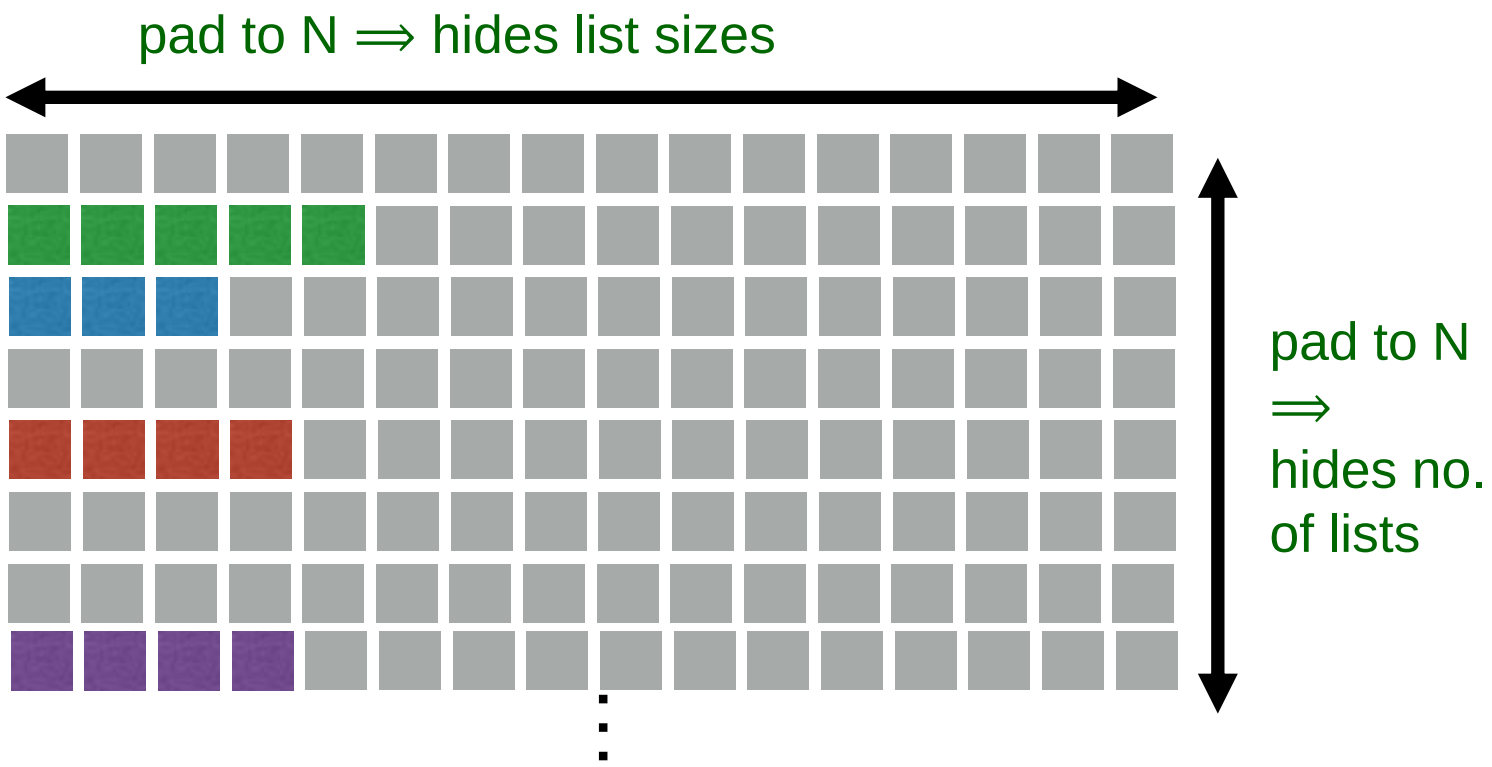
Index I_1

Soil	9,19,93,94,95
Plants	4,9,37
Flowers	9
Rose	9,15,42,75,78
Pots	9,37

1. Search $w_{1,1}$ = “Plants”
2. Search $w_{1,2}$ = “Soil”

Secure solution: Maximal Padding [CK]

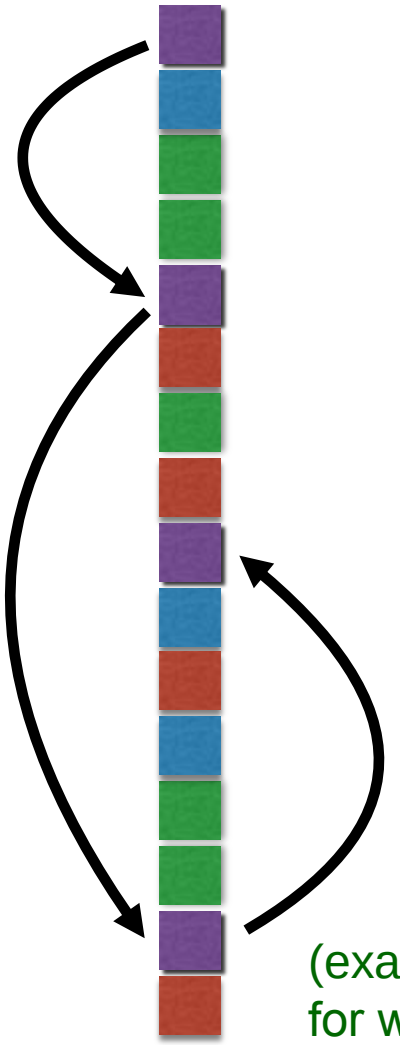
- pad all encrypted lists to size N
- store lists in rows in random order
- pad with extra dummy lists to hide # lists



Secure solution: Random Access Linked List [CGKO]

1. put ciphertexts in random order in array

2. link together lists with encrypted pointers



(example with poin
for word “Accept”)

Randomness vs. Structure

- We need randomness to avoid leakage to server

  … but we pay with wasted memory (padding solution)

  … or we pay with random access (linked list solution)

- Tradeoff: Random-access lists with multiple elements per entry
  - Each entry = fixed-size bucket (wasted space/read in half-filled buckets)

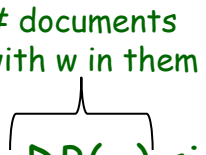
- Lower bound [CT'14]: Cannot be simultaneously optimal in:

  … locality, total space and goodput (read utilization)

- Asharov et al.: asymptotics close to optimal (can it be practical too?)

Packed Solution [NDSS'14*]

- Hybrid: Random-access lists with multiple elements per entry
  - Each entry = bucket
  - But single-size buckets don't work well with high-variable DB(w) sizes

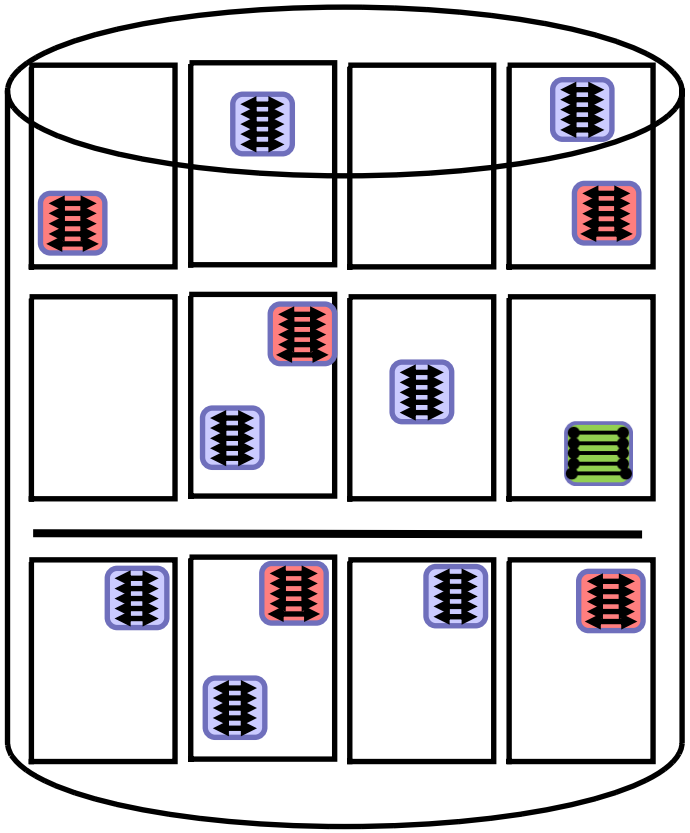


  - Hence we use a two-layer solution: buckets of pointers, each pointing to a block of identifiers
  - Plus: we use two-sized bucket* for optimizing goodput
- Pointer lists implemented w/history-independent dictionary structure Identifier blocks of arranged in an external array (*parallel access!*)
  - lists incur in storage allocation overhead but array does not

* NDSS'14 in eprint 2014/853; see C'13 eprint 2013/169 for a simpler scheme

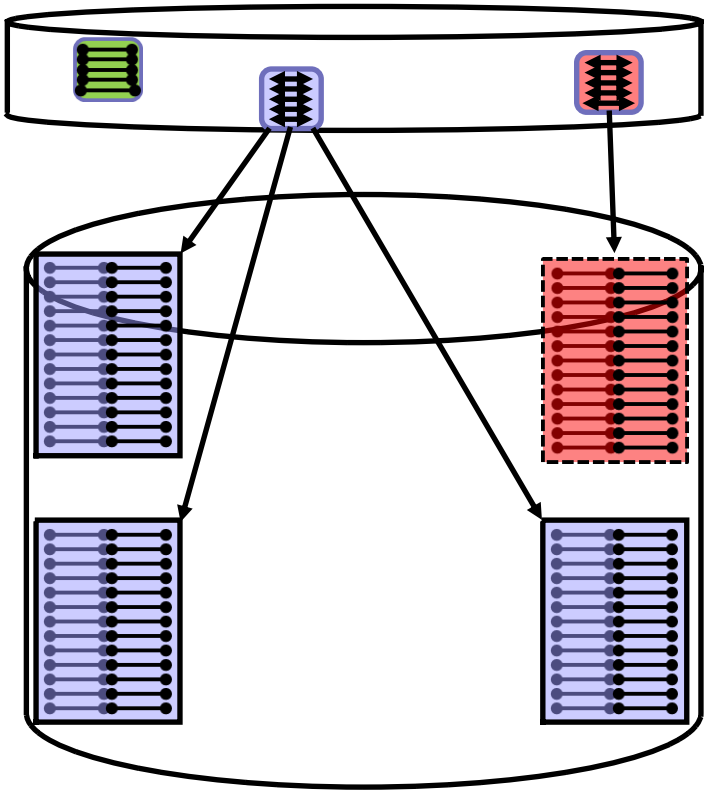
Faster Pre-Processing and Better Goodput

$\mathcal{T}_{pack}$: **Bucket (Paged) Hash (PH)**



- Low storage utilization (~60%)
- Cuckoo Hash fix (~90% util): sensitive to insertion history
- Low goodput

$\mathcal{T}_{2lev}$: **Two Levels (2L)**



- Multi-modal keyword distribution
- Good storage utilization (92%)
- High goodput.

LEAKAGE

Security Formalism (adversarial server)

- Based on the simulation-based definitions given for SKS [CGKO,CK].
- There is an attacker E (acting as the server), a simulator Sim and a *leakage function* L(DB, queries):
  - Real: Attacker E chooses DB and gets the pre-processed encrypted DB, then interacts with client on adaptively chosen queries
  - Ideal: Attacker E chooses DB and queries (adaptively), E gets Sim(L(DB)) and Sim(L(DB,queries))

  A SSE scheme is *semantically secure with leakage L* if for all attackers E, there is a simulator Sim such that the views of E in both experiments are indistinguishable

➔ Server learns nothing beyond the specified leakage L even if it knows (and even if it chooses *adaptively*) the plaintext DB and queries

SKS Leakage

- In all cases: Result set (matching enc'd documents) + query repetition
- Basic solution: Randomized linked list

  "Minimal" leakage: Only total number of record-keyword pairs

  $$\sum_{(w \in W)} |DB(w)|$$

- Better locality: Packed list (lists of blocks of size B)

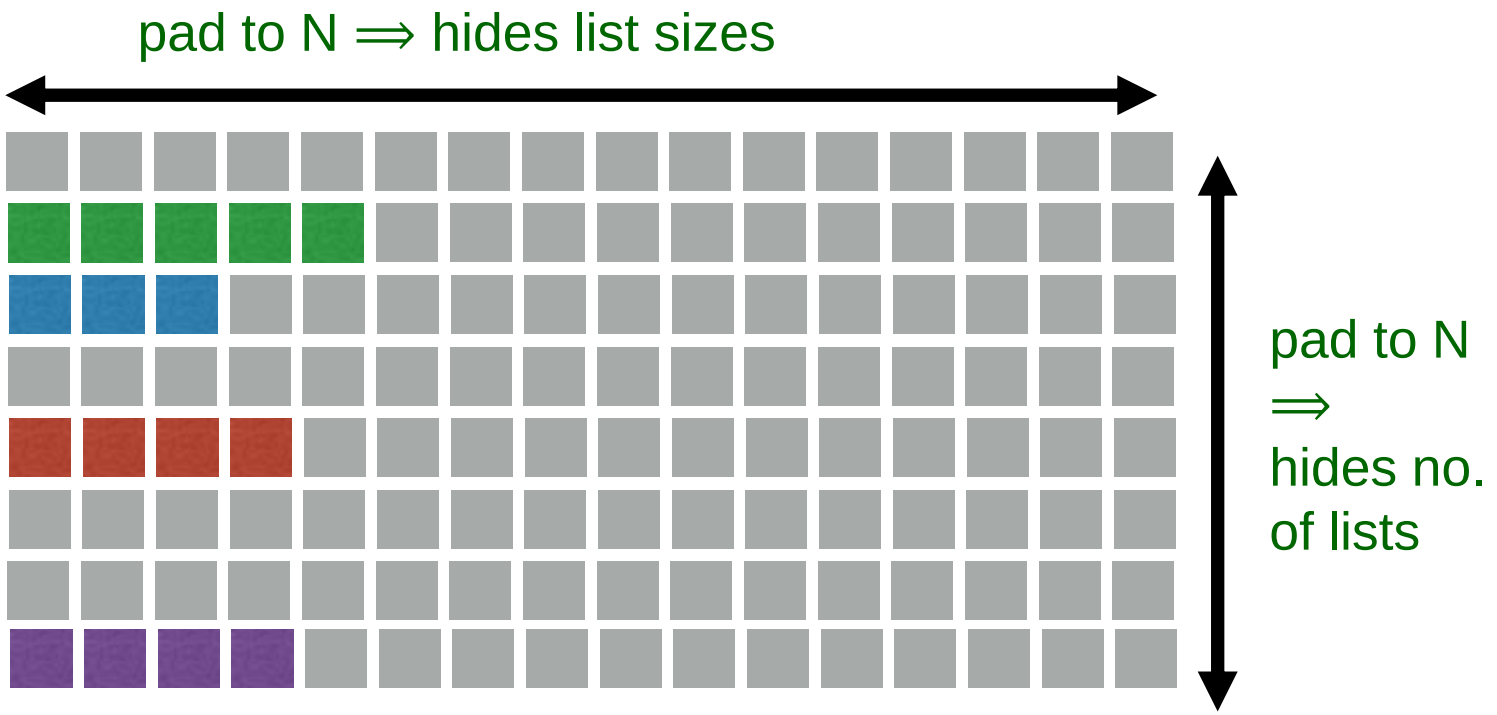
  Leakage: $\sum_{(w \in W)} \left\lceil \frac{|DB(w)|}{B} \right\rceil$ (incomparable leakage with above)

- 2-level implementation (blocks of b pointers, each to a block of size B

  Leakage = Size of external array (function of param's b and B)

- Adaptive security with ROM ("programmable PRF") or client interaction

Example for leakage analysis



Proving leakage (toy case)

- Consider a simple strategy where all w's are assigned equal-length arrays [Cash-Kamara]
  - Memory divided into N arrays, each of size M ; $N \geq |W|$, $M \geq |DB(w)|$, $\forall w \in W$
  - Each w in W is assigned one array (at random) which contains the permuted and encrypted set of all ind $\in$ DB(w).
  - Client processes query w by sending (K1,K2)=F(K,w) where K1 points to the w-array and K2 is used to decrypt each ind $\in$ DB(w).
- Leakage for queries $w_1,\ldots,w_n$ = $\{M, N, DB(w_i)\ i=1,\ldots,n, \{(i,j): w_i=w_j\}\}$
- Simulator (static): Given leakage profile: □ creates N arrays of size M;
  - chooses key K, and $\forall$ i, computes $(K1,K2)=F(K,w_i)$;
  - for ind $\in DB(w_i)$ (in permuted order) stores Enc(K2,ind) under array K1;
  - fills all other arrays with random values (assumes pseudorandom ciphertexts)

Proving adaptive security

- Same as above but
  - All arrays are initially filled in with random values
  - PRF replaced with RO which programs the DB(w) entries for each new query w

  OR
  - The decryption is done by the client sending a decryption pad for each ind in DB(w) – still a single message from client to server but of length |DB(w)|.

SKS Attacks [CGPR'15]

- See Sasha's presentation (IsraelWinterAB15Parts2,3.pptx #87…)
- Very basic attacks focusing on the following cases:
- Known documents and enc'd keywords ➔ learn per-keyword doc count which they claim is unique for a *large fraction of* keywords (use count to match b/w ptext and enc'd words, use intersection sizes for the rest)
- Unknown documents but encrypted keywords stored in the positions (w/repetitions) they appear in the document (substitution code)
  - Input to attack is a small number of representative ptext docs (w/enough words in them) and the corresponding encrypted keywords
- Previous item but keywords given w/o order, they use an active chosen-document attack
- Application to sound SKS (e.g. OXT):
  - First attack applies for actually queried keywords (if ptext DB is known);
  - last attack will apply only after most of the keywords were actually queried

Searchable Encryption:

OXT Protocol (Single Client Setting)

Hugo Krawczyk

IBM Research

Winter School – Bar Ilan University – January 2015

Terminology

- Client = Charlie C, Server = EDdie E
  - In the multi-client setting (covered later) DB owner called DeBbie
- DB = collection of plaintext records (aka documents) owned by client C
- EDB (encrypted DB): data stored at E (encrypted records + metadata)
- ind = index to plaintext records (whose order is randomized)
- DB(w)={ind_1, ind_2,…} indices of records containing w
- W(ind) = set of words contained in record ind,
- W = the set of all words in DB, i.e. $\cup_{ind} W(ind)$

Single-Keyword Search (SKS) in SSE

- SSE (searchable symmetric encryption)
  - A client C (both client and data owner) and server E
  - Client C transforms its plaintext DB into "encrypted DB" (EDB) that includes encrypted records and metadata;
  - EDB stored at server E; C only keeps a cryptographic key
- SKS: Given keyword w return *indices* of documents containing w
  - Important: *model simplified* by abstracting out retrieval (and enc/dec) of documents (note: variable vs. fixed-length ciphertexts)
- SKS at the basis of all our solutions

Conjunctive (and Boolean) Queries

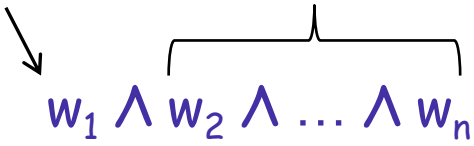
Cash et al, Crypto 2014, eprint 2013/169

Conjunctions: the naïve solution
(conj = $w_1 \wedge w_2 \wedge \ldots \wedge w_n$)

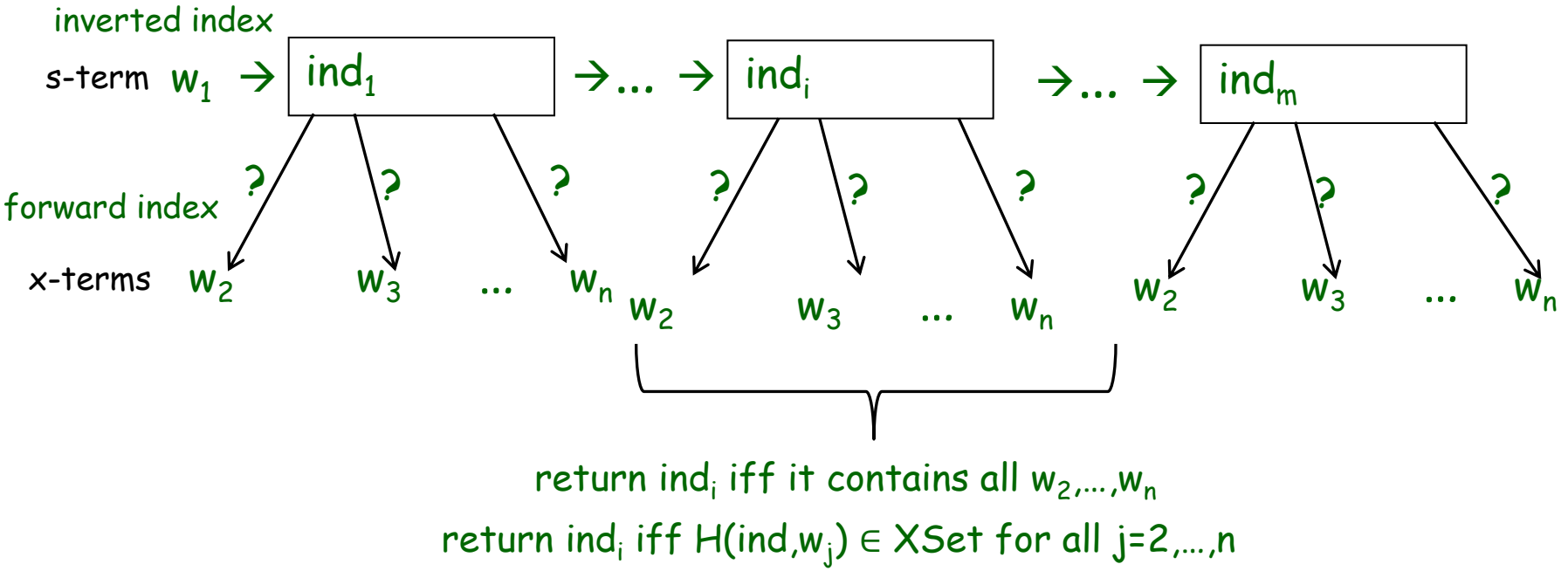
- Run single-keyword-search on each w_i to get ind's for each w_i, then retrieve ind's in intersection
- Performance: Work proportional to the sum of matching sets for each term, i.e. $|DB(w_1)|+\ldots+|DB(w_n)|$
  - Sum of costs of n single-term searches
  - Prohibitive with low entropy terms, e.g. "lastname=Jutla ∧ gender=male"
- Leakage: ind's for each $DB(w_i)$ leaked
  - Same effect as if each w_i was queried separately
    - union of leakages rather than their intersection
- Need to improve both cost and security…

Conjunctions via Forward Index (unencrypted)

- Inverted index: keyword w points to records containing w
  - $w \rightarrow DB(w) = \{ind_1, ind_2, \ldots, ind_t\}$
- Forward index: record points to all its keywords w
  - $ind \rightarrow W(ind) = \{w_1, \ldots, w_m\}$
- Conjunction algorithm (conj = $w_1 \wedge w_2 \wedge \ldots \wedge w_n$)
  - Let w_1 be the term with <u>smallest</u> $DB(w_i)$ set.
  - For each ind in $DB(w_1)$ and each w_j in conj, check if w_j in record ind.
    If tests succeeds for all w_j , j=2…n, return ind.
- Terminology: s-term vs x-terms



Resolving $w_1 \wedge w_2 \wedge \ldots \wedge w_n$ (w_1 least frequent) with forward indexing



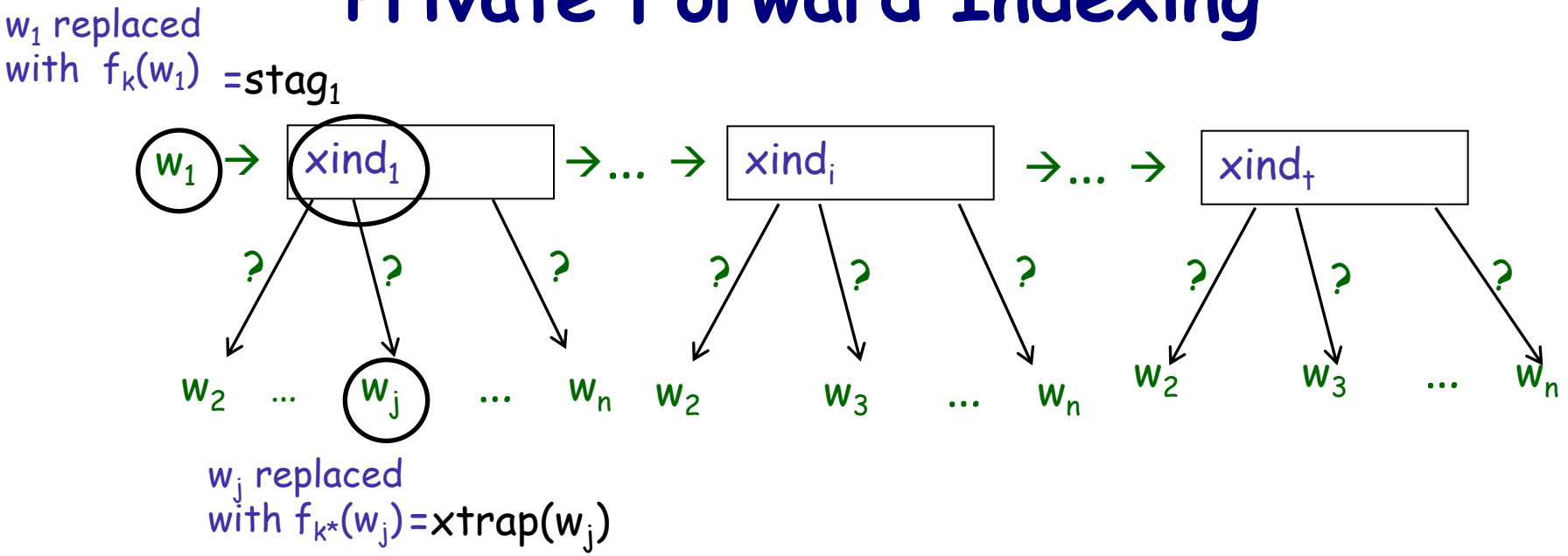
- Implementation trick: Build set XSet of hash values as follows

  For each record ind and each w in W(ind): add H(ind,w) to XSet

- To test if $w \in W(ind)$ check if $H(ind,w) \in XSet$

  ➔ $\forall$ ind $\in DB(w_1)$ return ind iff $H(ind,w_j) \in XSet$ for all j=2,…,n

Private Forward Indexing



- Implementation: $stag_1=f_k(w_1)$ $xtrap_j=f_{k^*}(w_j)$ $k^* \neq k$ two prf keys
- Preprocessing: ∀ xind and w in xind: add H(xind, xtrap(w)) to XSet
- Client hands to E: $stag_1$ and $xtrap_2, \ldots, xtrap_n$ → was H(ind,w)
- E returns record xind in $stag_1$ if for all j>1, H(xind, $xtrap_j$) ∈ Xset

(Basic Cross-Tag Protocol – BXT)

Computational cost of BXT

- E work is proportional to $|DB(w_1)|$: Major improvement over naïve sol'n
  - In naïve, cost is $|DB(w_1)| + |DB(w_2)| + \ldots + |DB(w_n)|$ (min vs max, e.g. gender=male)
  - Choosing w_1 right is important: Based on DB statistics
- "Non-interactive": Single *short* message from C to E and E sends back results

BXT Leakage

- Leakage to E: Substantial improvement over naïve solution – reduces correlation between x-terms across same or diff queries

  ☺ When w_i and w_j are x-terms in same or diff queries nothing is learned about $DB(w_i)$ and $DB(w_j)$ other than via intersections with s-terms

  😐 E learns repeated terms in different queries and learns the sizes of $DB(w_1)$ and of $DB(w_1 \wedge w_i)$ for each x-term w_i in the same query.

  😐 E learns ind's of all s-terms (➔ their size and intersections).

  ☹ Since E learns xtrap for each x-term w_i , it also learns $DB(w_1) \cap DB(w_i)$ for any pair of s-term w_1 and x-term w_i *even across queries*

- Next: How to reduce inter-query leakage. Avoid revealing ind's (inter s-term leakage) and xtrap's (inter x-term leakeage)

Terminology to remember

w_1 $w_2 \ldots w_n$

s-term x-terms

C: strap xtrap (trapdoors: for C only)

E: stag xtag (tags: revealed to E)

xtag = elements of Xset (i.e. xtag = H(xind,xtrap))

Tset(w) (tuple set = inverted index)

List $F_K(w) \rightarrow (Enc_{Kw}(ind_1), \ldots, Enc_{Kw}(ind_m))$ (SKS solution)

$Tset = \{Tset(w)\}_{w \in W}$

TsetSetup(T), TsetRetrieve(Tset,stag), stag←TsetGetTag(K_T,w)

Reducing cross-query leakage (via oblivious computation)

- Recall (BXT):
  - Client computes and hands to E: $stag_1$ and $xtrap_2$,…, $xtrap_n$
  - E returns record xind in $stag_1$ if for all j>1, H(xind, $xtrap_j$) ∈ Xset
- Goal: reduce cross-query correlations by hiding xtrap's from E
- Idea 1: Interactive protocol b/w C and E (IXT)
  - E sends each xind to C who sends back H(xind,$xtrap_j$), j=1,…,n
  - E learns H(xind,xtrap) but not xtrap and can check if conjunction holds for xind – all xtrap-related leakage avoided

☹ (i) Extra round of communication; (ii) E can cheat sending xind from diff. s-term (E learns forbidden intersections); (iii) leakage to client in MC-SSE (s-term inters.)
Yet, for HBC E and single-client SSE, IXT can be a great solution (w/ more latency)

Reducing cross-query leakage (via oblivious computation)

- Recall (BXT):
  - Client computes and hands to E: $stag_1$ and $xtrap_2$,…, $xtrap_n$
  - E returns record xind in $stag_1$ if for all j>1, H(xind, $xtrap_j$) ∈ Xset
- Goal: reduce cross-query correlations by hiding xtrap's from E
- Idea 2: Replace H(xind, $xtrap_j$) with *secure computation* (b/w E & C)
  - C inputs xtrap, E inputs xind
  - E learns H(xind,xtrap) but not xtrap; C learns nothing (e.g. xind)
  - Solution: $H(xind, xtrap) = xtrap^{xind} \bmod p$

    (Group $G=\langle g \rangle$ of prime order p, xtrap ∈ G, xind ∈ Z_p)

Interactive Solution

- $H(xind, xtrap) = xtrap^{xind} \bmod p$
- E has "key" xind, C has xtrap
  - C to E: $a \leftarrow xtrap^{z}$ for z random in Z_p
  - E to C: $b \leftarrow a^{xind}$
  - C to E: $xtag \leftarrow b^{1/z}$ $(= xtrap^{xind})$
  - E searches xtag in XSet
- Problems:
  1. E knows xind hence it learns xtrap $(=xtag^{1/xind})$ (didn't gain much)
  2. E learns xind hence it learns relation b/w s-terms (e.g. $|DB(s) \cap DB(s')|$)
  3. Rounds of interaction

Non-Interactive Solution

- Interactive: E has key xind (from stag tuple), C has xtrap (from DB)
  - C to E: $a \leftarrow xtrap^z$ z random in Z_p
  - E to C: $b \leftarrow a^{xind}$
  - C to E: $xtag \leftarrow b^{1/z}$ $(= (xtrap^z)^{xind/z} = xtrap^{xind})$

- Non-interactive: *store xind blinded with a one-time blinding factor z*
  1. At setup: $y = xind/z$ is stored at EDB;
  2. At search C sends $a = xtrap^z$ to E (C derives xtrap from w_j and z from w_1)

     E *retrieves* y from EDB and sets $xtag \leftarrow a^y$ $(= (xtrap^z)^{xind/z} = xtrap^{xind})$

     ↓ avoids interaction (and prevents E from learning xtrap or xind!)

OXT: Oblivious X-Tags Protocol

- Non-interactive: store xind blinded by a one-time blinding factor z
  1. At setup: $y=xind \cdot z^{-1}$ is stored at EDB;
  2. At search C sends $a=xtrap^z$ to E (C computes xtrap from w_j and z from w_1)
  3. E retrieves y from EDB and sets $xtag \leftarrow a^y$
- OXT basics, 2-term conjunction example (w_1,w_2)
  - EDB setup: $\forall$ w in W, for t=1 .. $T_w=|DB(w)|$
    - store $y_t=xind \cdot z_t^{-1}$ in w-list with corresp. encrypted ind ($z_t=F(w,t) \in Z_p$)
  - On query (w_1,w_2): C computes xtrap (a prf applied to w_2)
    - C sends to E $stag(w_1)$ and the T_{w1}-long vector $\{a_t=xtrap^{z_t} : t=1,\ldots,T_{w1}\}$
  - For t=1..T_{w1}, E retrieves y_t from stored w_1-list, sets $xtag_t \leftarrow a_t^{y_t}$ (= $xtrap^{xind}$)
  - E returns t-th encrypted ind iff $xtag_t$ in Xset

OXT Core

- E Setup: For all w in W:
  - $strap_w = F(K_S, w)$; $xtrap_w = g^{F(K_x,w)}$
  - For t=1 to T=|DB(w)|:
    - $Tset(t) = [\ Enc(K_e, ind_t),\ y_t = xind_t \cdot z_t^{-1}\ ]$ (*
    - Add $xtag = (xtrap_w)^{xind}$ to Xset (* where $xtrap_w = g^{F(K_x,w)}$ *)

Preprocessing: #(w,ind) expon's*

Search: Per item in DB(w_1):

- Client n-1 expon's*
- Server: n-1 expon's

* same-base

- Search on $(w_1, w_2, \ldots, w_n)$:
  - C computes (using keys K_S, K_X): $strap_1, xtrap_2, \ldots, xtrap_n$
  - For t=1…|DB(w_1)|, C sends to E: $\{x_j = xtrap_j^{Zt},\ j=2,\ldots,n\}$ (* $z_t = F(strap_w, t)$ *)
  - For t=1…|DB(w_1)|, E sets $xtag_{j,t} = x_j^{y_t}$, j=2,…,n (* y_t stored in Tset *)
  - E returns t-th encrypted ind iff for all j=2,…,n, $xtag_{j,t}$ in Xset
- It works because $xtag_{j,t} = x_j^{y} = (xtrap_j^{z})^{y} = (xtrap_j^{z})^{xind/z} = xtrap_j^{xind}$

OXT Leakage (Improvements on BXT)

- Repetition of s-term still visible to E (stag is deterministic) but x-term repetition mostly* avoided (see below).
- Most important: OXT solves the bad inter-query leakage where

  ☹ E learns intersection $DB(w_1) \cap DB(w_i)$ for any pair of s-term w_1 and x-term w_i *even across queries*

  (in OXT can't combine x-term from one query with an s-term from another)

- The following milder leakage remains:
  - For queries $w_1 \wedge x$ and $w_1' \wedge x'$, if $DB(w_1 \wedge w_1') \neq \emptyset$ and $x=x'$ then E learns that $x=x'$ and the encrypted ind's in $DB(w_1 \wedge w_1')$
  - Leakage unlikely for s-terms chosen as low-frequent terms ($w_1 \wedge w_1'$ would be usually empty); and it is impossible if both w_1, w_1' are, say, last names.

Summary: OXT Leakage to E

- As in BXT:
  - Total index size = upper bound on Σ_i $|DB(w_i)|$ (Tset leakage)
  - Number of terms in each conjunction
  - Size of s-term set $|DB(w_1)|$ (unavoidable? Reduction from 3SUM)
  - s-term repetitions
  - Encrypted ind's in the set $DB(w_1 \wedge w_j)$, j=2,…, n (e.g. $|DB(w_1 \wedge w_j)|$)
  - NO leakage about intersections of x-terms in same or different queries
- Improvement on BXT:
  - For queries $w_1 \wedge x$ and $w_1' \wedge x'$, if x=x' and $DB(w_1 \wedge w_1') \neq \emptyset$, then E learns that x=x' and the encrypted ind's in $DB(w_1 \wedge w_1')$
- Server E can be malicious but trusted to return the correct results

OXT non-leakage

- OXT *does not* reveal
  - plaintext data (semantically secured, no det'c enc, no repeated patterns, etc)
  - plaintext queried values (s-term and x-terms)
  - plaintext ind's other than those matching the conjunction
  - information on intersecting records of different x-terms (in the same query or across different queries)
  - repeated x-terms or intersections between different s-terms, except for those revealed via last leakage item in previous slide
    - This leakage can be reduced substantially with more memory (practical for moderate number of keywords per document, e.g. 100 keywords/record)
      - An example of space-privacy trade-offs

OXT Theorem

- Security formalism: Same simulation-based definition as in SKS
- Let L be (a formal description of) the leakage function described before.
- Theorem: OXT is semantically secure with leakage L under the DDH assumption when implemented with a secure PRF and CPA encryption.
- Proof: see paper for painful enjoyment… eprint 2013/169
- Note: No ROM required!
  - ROM used in our implementation of the more advanced models and to improve communication in the adaptive security case.

Simulation Ideas

(details in eprint 2013/169)

- Tset simulation (as in SKS)
- y values (=xind z^{-1}) chosen as random elements of Zp
- Tricky part: How to choose Xset values so that $(xtrap_w)^{xind} \in$ Xset iff w is in record xind. Note that:
  - (i) any w can be queried (even if not in DB);
  - (ii) the values $x=xtrap^{1/z}$ sent by the client and the stored random y need to satisfy that x^y is (or is not) in Xset depending on whether "w ∈ ind";
  - (iii) x^y values may repeat for different values of x and y
  - Solution: Choose Xset values as random elements h in group G; simulate client values $xtrap^{1/z}$ as $h^{1/y}$ depending on retrieved y (so that $(xtrap^{1/z})^y = h$)
    - Use DDH to claim that random Xset values are indistinguishable from real (structured) $g^{w\ xind}$ (note that w and xind may repeat in multiple Xset entries)

General Boolean Formulas

- Queries of the form $w_1 \wedge \Phi(w_2,\ldots,w_t)$ where Φ is any Boolean formula (program) → SNF = "Searchable Normal Form"
- Similar to the conjunctions mechanism:
  - For all ind $\in$ DB(w_1), set β_j=1 if H(ind,w_j) $\in$ Xset, return ind iff $\Phi(\beta_2,\ldots,\beta_t)$ = true
  - Same cost as for conjunction
  - Any Boolean query via "w_1 = True" (linear in worse case)
- More generally: Disjunction of any number of such formulas
- Example: (m out of t)-threshold query ⇨ disjunction of (t-m+1) formulas:

  $(w_1 \wedge T_{(m-1,t-1)}(w_2,\ldots,w_t))$ or $(w_2 \wedge T_{(m-1,t-2)}(w_3,\ldots,w_t))$ or … or $(w_{t-m+1} \wedge T_{(m-1,\ldots)}(w_{t-m+2},\ldots,w_t))$
- Leakage:
  - As in conjunctive search, plus EDB learns Φ (not w_i's) and the bits β_j for j=2,…,t

Extensions

- Dynamic DBs: additions, deletions, modifications (and client caching)
- More complex query types
  - Range queries: return all records with DOB between 2/3/87 and 3/4/88
  - Substring/Wildcard queries: %lope%, enc_clo_ _dia, %cycl_ _ _dia
  - Phrase queries: "searchable symmetric encryption", "Gone _ _ Wind"
- PXT: Very communication-efficient (as in BXT: short client message) but uses pairings, same leakage as OXT
- Complex operational/trust settings: MC-SSE and PIR-SSE:
  - Malicious clients, hiding queries from Debbie (and even hiding policy as in warrant-based scenarios)
  - Tools (all via simple exponentiation): OPRF (for PIR), attribute-based keys for blind authorization, homomorphic signatures for query authentication by Eddie

Updates

Dynamic Data (updates)

- Updates: add, delete, modify
  - Operational assumption: |DB|>>updates; thus:
    - EDB super-optimized for disk access but update structures planned for RAM
    - Periodic re-encryption eliminates leakage trails (e.g. new/old records)
- Caching (defense against leakage)
  - Client can identify previously retrieved documents in result set before requesting them
  - Will retrieve a previously retrieved document only if the document changed since last retrieval
  - Important defense against server learning intersection b/w queries
    - but leakage on the number of matching-but-not-retrieved documents

Data Structures (NDSS'14, eprint 2014/853)

- EDB (Tsets + Xsets) unchanged
  - Changes to clear-text DB do not affect EDB
- EDB+ records DB changes between re-processing phases
  - RAM resident Hash Tables (dictionaries)
  - TSet+ stores new tuples
  - XSet+ stores new XTags
  - RevID Set stores *revocation* IDs for deleted records
- Clear-text database enhanced
  - |Tset(w)|, for all keywords w
  - Seq # of last successful update

Integrating EDB+ into OXT

- Eddie runs OXT on EDB first and on EDB+ next
  - EDB+ tuples are labeled separately (not grouped in blocks)
  - OXT result set filtered using the RevID Set
  - Very efficient
- Charlie and Debbie are unchanged
- Security
  - Cannot relate old and new tuples *except* if keyword was searched (can avoid with evolving periodic keys)
    - Can hide operation (add/del/modify) by always doing a delete+add
  - But Eddie knows if returned record (and touched tuple) is new or old
- Caching: Can identify previously cached records and know if they changed

Complex Queries
(summary in case I don't get to do them in detail)

Extensions to OXT – reductions to Boolean queries

Range Queries

- E.g., return all records with DOB between 2/3/87 and 3/4/88
- Compute a cover of the range by intervals (via a tree cover algorithm)
- Generate a disjunction of values representing each of the intervals
  - Range R translated into disjunction of up to 2 log R exact-match terms
- Thus: the Boolean OXT protocol applies AS IS
- The more interesting part: Blind authorization
  - Debbie authorizes based on total size of range (eg. query span ≤ one year)
  - Needs to let Debbie learn the total size of the range w/o leaking on the endpoints? (e.g.., # of intervals should be same for any two ranges of same size)
  - Two solutions: Canonical covers (all ranges of given size have same-lengths intervals); 3-node over-covers (always 3 intervals with 40% avg overhead)

Substring/Wildcard Queries

- Any combination of _ ("single character wildcard") and substrings of k or more consecutive characters (k is a tunable parameter)
- Plus a % ("any string") at the beginning and/or end of a search expression
- Examples (return "encyclopedia"):
  (i) %cycl_ _ _dia (ii) %lope% (iii) enc_clo_ _dia
  - Tunable k: smaller for more general queries; larger for search efficiency
  - Variable k: flexibility and efficiency (only "anchor" ≥ k)

OXT Protocol (substring extensions)

- Substrings: we treat k-gram's as keywords, associated to a xind as well as a position p in xind – we thus extend the function H to:

$$H(\text{"cha"}, \text{xind}, p) = (\text{prf}(\text{"cha"}))^{(\text{xind})^p}$$

- We want to know if "charan" is in record xind:
  - We find (encrypted) values of y=xind and v=xind^p in Tset("cha") (which means: "cha" is in pos p in xind),
  - We then check if "ran" is in position p+3 in xind by checking that

$$(\text{prf}(\text{"ran"}))^{v\,y^3} = (\text{prf}(\text{"ran"})^{(\text{xind})^{P+3}} \quad \text{in Xset}$$

- Computed via a non-interactive 2-party secure function evaluation: Charlie has ("ran",Δ), Eddie has encrypted (y,v) and Δ

Generalization: Proximity Queries

- A generalization of our substring technique
- Can do search of the form (e_1,e_2,Δ) meaning
  - Return all records where element e_1 is at distance Δ from e_2 (Δ can be negative)
- Examples:
  - e_i are k-grams: resolves substrings and wildcards
  - e_i are textual words: resolves phrases (e.g., "Bar Ilan University")
  - Multi-dimensional distances (e.g., grid), etc.

Subsequences Leakage to Eddie

Reminder: Leakage from conjunctions $w_1 \wedge \ldots \wedge w_n$

1. Index size = upper bound on Σ_i $|Doc(w_i)|$
2. Number of terms in each conjunction
3. Size of s-term set $|Rec(w_1)|$ and whether s-term repeats
4. Size of $Rec(w_1 \wedge w_j)$, $j=2,\ldots, n$
5. For queries $w_1 \wedge x$ and $w_1' \wedge x'$, if $x=x'$ and $Rec(w_1 \wedge w_1') \neq \phi$, then E learns that $x=x'$ and the encrypted rind's in $Rec(w_1 \wedge w_1')$

Leakage is similar for subsequence queries with s-term k-gram w_1 and x-term grams $w_2,..,w_n$ except for more involved 5' (stated for 2-term query)

- For queries $w_1 \wedge x$ and $w_1' \wedge x'$, with offsets Δ_1, Δ_2 , if $x=x'$ and there exist ind in $Rec(w_1 \wedge w_1')$ and p, p' such that w_1 is in position p in ind and w_1' in position p' in ind, and $p-p'= \Delta_1- \Delta_2$ then E learns that $x=x'$, and the encrypted pairs (ind,p), (ind,p').
- Plus: offset Δ leaks – can be avoided with a round of communication C-E

Leakage from Range Queries

For Debbie:

- The total size of the queried range (necessary to apply policy)

For Eddie:

- Leakage for atomic range query with cover $w_1=(h_1,c_1)$ … , $w_n=(h_n,c_n)$ is same as for OXT disjunction "w_1 or … or w_n"
  - $|DB(w_i)|$ for i=1,…,n , $DB(w_1$ or … or $w_n)$
- For composite queries, leakage is same as OXT where the range query is replaced with "w_1 or … or w_n"

For Client:

- Mask($|DB(w_i)|$) for i=1,…,n if range query acts as s-term

Searchable Encryption:

Multi-client and OSPIR Settings

Hugo Krawczyk

IBM Research

Winter School – Bar Ilan University – January 2015

Encrypted Search I (SSE)

- Owner of database DB (= client) *outsources* its Encrypted Data to a server (EDdie) such that:
- Owner/Client:
  - pre-processes data, outsources to Eddie, keeps only a cryptographic key, later runs queries at Eddie, retrieves/decrypts matching documents
- Eddie:
  - gets all DB documents in encrypted form
  - keeps index information (metadata) in encrypted form
  - responds to client's queries (returns matching encrypted doc's)
  - does not learn the searched terms or DB plaintext information
    - but leakage on data-access patterns and query patterns allowed

Encrypted Search II (Multi-Client SSE)

- Owner of DB (DeBbie) outsources DB to Eddie such that Eddie (as before):
  - keeps all records and index information in encrypted form
  - can accurately respond to any boolean query (returning matching records)
  - does not learn the searched terms or any plaintext information on the DB (some leakage allowed)
- While Debbie:
  - *can delegate search to clients* (via search tokens)
  - such that clients can search through queries authorized by Debbie but learn nothing about data not matching the authorized queries
  - multiple and adversarial clients (fully malicious in our solutions)

Encrypted Search III (OSPIR=Outsourced Symmetric PIR)

- As scenario II

  PLUS

- Debbie can authorize clients to perform queries according to a prescribed policy

  (i.e., determine the query compliance and provide the corresponding tokens)

- … but she has to do so without learning the searched terms

  - Assumption: Debbie and Eddie do not collude (otherwise the strong performance limitations of PIR apply)

The MC-SSE Setting

D gives the tokens to C and authorizes according to a policy

(leakage to C: anything beyond the results to authorized queries)

OXT Core

- EDB Setup: For all w in W:
  - $strap_w = F(K_S, w)$; $xtrap_w = g^{F(K_x, w)}$
  - For t=1 to T=|DB(w)|:
    - $Tset(t) = [\ Enc(K_e, ind_t),\ y_t = xind_t \cdot z_t^{-1}\ ]$ (* where $z_t = F(strap_w, t)$ *)
    - Add $xtag = (xtrap_w)^{xind}$ to Xset (* where $xtrap_w = g^{F(K_x, w)}$ *)
- Search on $(w_1, w_2, \ldots, w_n)$:
  - C computes (using keys K_S, K_X): $strap_1, xtrap_2, \ldots, xtrap_n$
  - For t=1..T, C sends to E: $\{x_j = xtrap_j^{Z_t},\ j=2,\ldots,n\}$ (* where $z_t = F(strap_w, t)$ *)
  - For t=1..T, E sets $xtag_{j,t} = x_j^{y_t}$, j=2,…,n (* y_t stored in Tset *)
  - E returns t-th encrypted ind iff for all j=2,…,n, $xtag_{j,t}$ in Xset
- It works because $xtag_{j,t} = x_j^{y} = (xtrap_j^{z})^{y} = (xtrap_j^{z})^{xind/z} = xtrap_j^{xind}$

MC-OXT

- In OXT: Search on $(w_1, w_2, \ldots, w_n)$:
  - C computes (using keys K_S, K_X): $strap_1$, $xtrap_2, \ldots, xtrap_n$
  - For t=1…$|DB(w_1)|$, C sends to E: $\{x_j = xtrap_j^{Z_t}, j=2,\ldots,n\}$ (* $z_t = F(strap_w, t)$ *)
  - For t=1…$|DB(w_1)|$, E sets $xtag_{j,t} = x_j^{y_t}$, j=2,…,n (* y_t stored in Tset *)
- Adapting to the MC-SSE setting – initial ideas:
  - D (using keys K_S, K_X) provides C with $strap_1$, $xtrap_2, \ldots, xtrap_n$
    - Fails: C can combine strap from one query with xtrap's from another to obtain an unauthorized query
  - Solution: D signs $(strap_1, xtrap_2, \ldots, xtrap_n)$ so that E can verify binding
    - Fails: C does not pass xtrap values to E but rather $xtrap^z$ and revealing z values to E is insecure: allows E to do unauthorized searches (back to BXT)

  ➔ D needs to sign $(strap_1, xtrap_2^z, \ldots, xtrap_n^z)$ for many z's (but how many?)

MC-OXT

- In OXT: Search on $(w_1,w_2,\ldots,w_n)$:
  - C computes (using keys K_S, K_X): $strap_1$, $xtrap_2,\ldots,xtrap_n$
  - For t=1..T, C sends to E: $\{x_j=xtrap_j^{Zt},\ j=2,\ldots,n\}$ (* where $z_t= F(strap_w, t)$ *)
  - For t=1..T, E sets $xtag_{j,t} = x_j^{y_t}$, j=2,…,n (* y_t stored in Tset *)
- Adapting to the MC-SSE setting ("homomorphic signature"):
  - D needs to sign $(strap_1, xtrap_2^z,\ldots,xtrap_n^z)$ for many z's (but how many?)
  - Solution: D provides C with $(strap_1, xtrap_2^{r2},\ldots,xtrap_n^{rn})$ for random $r_2,\ldots,r_n$, and also $AuthEnc(K_M; r_2,\ldots,r_n)$ where K_M is key shared between D and E.
  - C will send $x_j=(xtrap_j^{rj})^{Zt}$ (* instead of $xtrap_j^{Zt}$ *)
  - E decrypts and verifies $r_2,\ldots,r_n$, then it computes $xtag_{j,t} = x_j^{yt/rj}$
    - Note: E does not verify the signature on x_j's, but by raising to the r_j^{-1} it ensures that if C is cheating, the xtag will result in a random value (w.h.p not in Xset)

The OSPIR Setting

As in MC-SSE but D authorizes queries according to a policy without learning the queried values (a la PIR*)

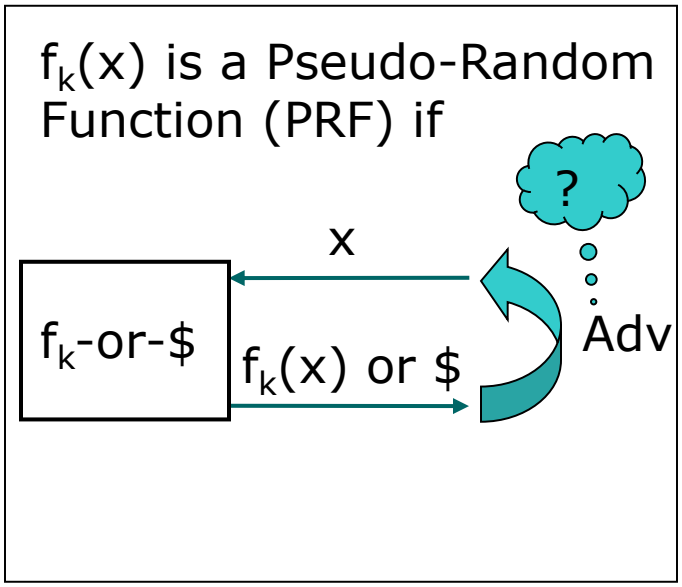
OSPIR = "Outsourced Symmetric PIR"

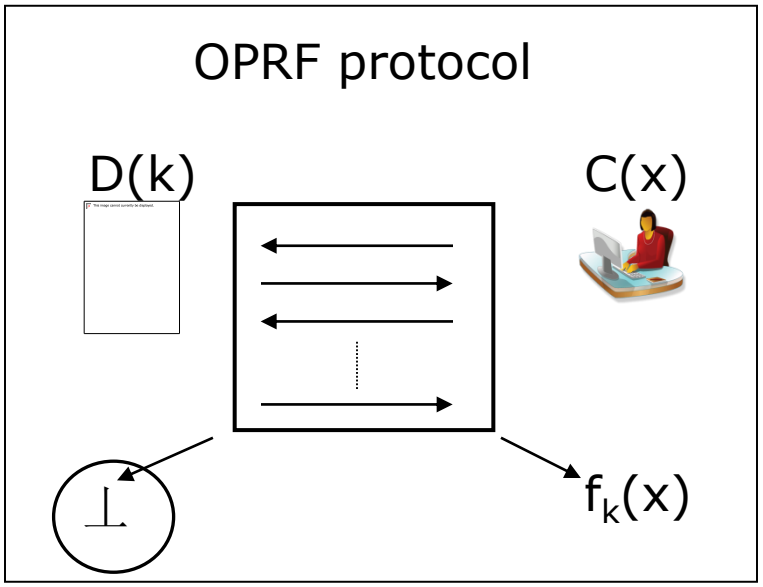
(*PIR = Private Information Retrieval CGKS'95)

Multi-Client SSE with Blind Authorization (OSPIR)

- We call this setting "outsourced symmetric PIR (OSPIR)"
- Parties: Client C, DB owner D (authorizer), EDB holder E
- Keywords are attribute-value pairs, e.g. ("name",Joe), ("text", I am happy)
- Attribute-based policies ("Is client C authorized for query Q?")
  - Policy decisions based on attributes not values
    - E.g. can query name and lastname but only with one of (zipcode, town, school)
    - Permissions set by D and can depend on client and type of boolean query
- D enforces policy w/o learning the queried values, only the attrib's
  - or less, e.g. a class of attrib's a term belongs to, not the specific attrib

Basic Tool: Oblivious PRF (OPRF) [NR'04,FIPR'05]





- OPRF Instantiation: $f_k(x)=[H(x)]^k$ (DH OPRF)
- Oblivious computation via "Blind DH Computation":
  - C sends $a = [H(x)]^r$ to D, D replies with $b = a^k$, C computes $f_k(x)$ as $b^{1/r}$

Warm-Up: Single-Keyword Search

- DB: Collection of inverted indexes pointed by each keyword (i,val)
  - (i,val) → {list of doc's containing (i,val)}
- EDB: Collection of inverted indexes using PRF-computed pointers (for hiding the keyword from Eddie)
  - f_K(i,val) → { encrypted list of records containing (i,val) }
- Policy: For each client C, Debbie has a list A_C of allowed attributes (i.e., C can search for any (i,val) such that $i \in A_C$)
- Case 1: Debbie is allowed to learn the query
  - 1. C → D: (i, val) 2. D → C: if $i \in A_C$ then return f_K(i,val)
  - 3. C → E: f_K(i,val) 4. E → C: Records pointed by f_K(i,val)

Single-keyword / Query hidden from Debbie

- Case 1: Debbie learns query (f_K = PRF)
  - 1. C → D: (i, val) 2. D → C: if $i \in A_C$ then return $f_K(i,val)$
  - 3. C → E: $f_K(i,val)$ 4. E → C: Records pointed by $f_K(i,val)$
- Case 2: Debbie learns attribute i but not value val
  - Replace PRF f_k with *Oblivious PRF* f_k:
    - D enters k, C enters (i,val), C learns $f_K(i,val)$, D only learns i
  - But how does D know if attribute i was authorized for C?
    - C can disclose i but then how does D knows that input (i,val) has same i?
    - Need a "conditional OPRF" (return output to C only if $i \in A_C$)
  - Simple solution: per-attribute OPRF key K_i: C learns $F_{K_i}(i,val)$
- If C claims attrib i but enters (j,val), he learns $F_{K_i}(j,val)$ which will return nothing at Eddie (e.g. zipcode=michael)

Conjunctions Case

- Given (i_1,v_1), (i_2,v_2),…,(i_n,v_n), return all records containing all these words
- Policy: A_C = subsets of {1,…,N}, e.g. if {1,3,8} ∈ A_C then C is allowed a conjunction of the form $(1,v_1)$, $(3,v_2)$, $(8,v_3)$ for any v_1, v_2, v_3
  (can have more compact representations, e.g. any 2-out-of-{1,3,8,11})
- Extension from the single-keyword case (example (i,u) ∧ (j,v))
  - D provides C with $F_{Ki}(i,u)$ and $F_{Kj}(j,v)$ via OPRF
  - But then C can combine two allowed queries into a non-compliant one
    - Given pair $F_{Ki}(i,u)$, $F_{Kj}(j,v)$ and pair $F_{Ki'}(i',u')$, $F_{Kj'}(j',v')$, C can query (i,u) ∧ (j',v')
- Solution: Let D sign the tokens $F_{Ki}(i,u)$, $F_{Kj}(j,v)$ given to C, Eddie will verify the signature before serving the query
  - But how can D sign OPRF output values she does not (and should not) know?

Signing tokens against *mix & match*

- Solution via "homomorphic signatures"
  - Exploit the homomorphic properties of the DH OPRF

Recall DH OPRF

- Cyclic group G of prime order q; H hash function from $\{0,1\}^*$ to G
- OPRF: $Z_q \times \{0,1\}^* \rightarrow G$, $F_K(w) = H(w)^k$
- Two party computation of $F_K(w)$: (* similar to a blind signature *)
  - D has key K in Z_q , C has input w in $\{0,1\}^*$
  - C to D: $a=H(w)^b$ for b random in Z_q
  - D to C: $c=a^k$
  - C: $F_K(w) \leftarrow c^{1/b}$ (* = $(((H(w)^b)^K)^{1/b} = (H(w))^K$ *)

Signing tokens against *mix & match*

- Solution via "homomorphic signatures"
  - Exploit the homomorphic properties of the DH OPRF
- C → D: $a_1 = (H(i_1,v_1))^{b1}$, $a_2 = (H(i_2,v_2))^{b2}$ (b_1,b_2 random in Z_q)
- D → C: $c_1=a_1^{Ki1\cdot r1}$, $c_2=a_2^{Ki2\cdot r2}$ (r_1,r_2 random in Z_q)

  env = $AEnc_{DE}(r_1, r_2)$ (* AEnc key shared between D & E *)
- C → E: $(H(i_1,v_1))^{Ki1\cdot r1}$, $(H(i_2,v_2))^{Ki2\cdot r2}$, env (*C de-blinds by raising to 1/b *)
- E: Verifies and decrypts r_1, r_2, computes $(H(i_1,v_1))^{Ki1}$, $(H(i_2,v_2))^{Ki2}$ and serves the query
  - To mix $(H(i_1,v_1))^{Ki1\cdot r1}$, $(H(i_2,v_2))^{Ki2\cdot r2}$ with $(H(i_1',v_1'))^{Ki1'\cdot r1'}$, $(H(i_2',v_2'))^{Ki2\cdot r2'}$, C would need to forge env = $AuthEnc_{DE}(r_1, r_2')$.
  - *Otherwise, if C uses a valid env, E derives random values not in EDB.*

Cost and Extension to Boolean Queries

- Authorization mechanism *very* cheap: one round of communication, 2n+1 exponentiations for the client and n+1 for Debbie (on n terms)
  - Base SSE protocol (OXT) already uses exponentiations for search, much more intensively and very optimized

- Boolean queries: Same as conjunctions but env includes description of expression ϕ (query type) plus "signatures" $r_1,\ldots,r_n$
  - E.g. "x1 and (not x2 or x3)", r1, r2, r3

symbolic expression

Security

- OSPIR-OXT leakage
  - To D: Query type and input attributes (values are info-theoretic protected)
  - To C: Size of s-term (or an upper bound if E sends dummy values - *unavoidable*)
  - To E: No extra leakage relative to basic OXT
- Security proven against malicious clients
  - I.e., no behavior by clients (even collusion between multiple clients) can lead to authorization of non-compliant queries or to learning policy
    - assumes "one-more DH" and ROM for OPRF implementation
- … and malicious Debbie, but *assumes non-collusions with E*
  - No behavior by Debbie can lead to learn information on queried values
- Note: Can add replay protection to env (one-time use, exp. date, etc.)

Authorization Extensions/Enhancements

- Debbie learns class of attributes, not individual attributes
  - E.g.: Debbie authorizes any conjunction with attribute 1 and any attribute from {2,3,4}, then Debbie *does not need to learn* which of 2,3,4 used
  - Solution: Debbie raises $H(i_1,v_1)$ to K_1 and $H(i_2,v_2)$ to K_2,K_3,K_4, C chooses one
- Role of Debbie can be split:
  - Holder of plaintext DB generates EDB; outsources EDB to Eddie and delegates the per-attribute authorization keys to Authorizer
    - The former needs not know the policy, the latter does not need DB
- Policy Manager: A 3rd party that holds policy, authorizes queries, but can't provide search tokens without Debbie's participation ("warrant")

Searchable Encryption:

Extending the OXT Protocol with Substring and Range Queries

Hugo Krawczyk

IBM Research

Winter School – Bar Ilan University – January 2015

Substring Search

Idea: Represent substring as conjunction of k-grams

- Preprocessing: Tag each text by all its 3-grams (k=3)

  e.g. Charanjit ⇨ "cha" , "har" , "ara" , "ran" , "anj" , "nji" , "jit"

- Search by Substring:
  - Search on a conjunction of all 3-grams in the substring

  e.g. *charan* ⇨ "cha" & "har" & "ara" & "ran"

Problem: False Positives

e.g. Search on ("cha" & "har" & "ara" & "ran") returns:

"... Harry chased the oranges rolling around in his garage ..."

Substring Search

Idea: Represent substring as conjunction of k-grams

Problem: False Positives

Refinement: Account for k-gram positions

- Preprocessing: Tag each text by (3-grams,position) pairs

e.g. Charanjit ⇨ (1,"cha"), (2,"har"), (3,"ara"), (4,"ran"), (5,"anj"), (6,"nji"), (7,"jit")

- Search by conjunction of (3-gram,shift) pairs

e.g. *charan* ⇨ (0,"cha") & (~~1,"har") & (2,"ara")~~ & (3,"ran"))

Problem:
- Positions in DB keywords are absolute
- Positions in query are relative

Substring Search

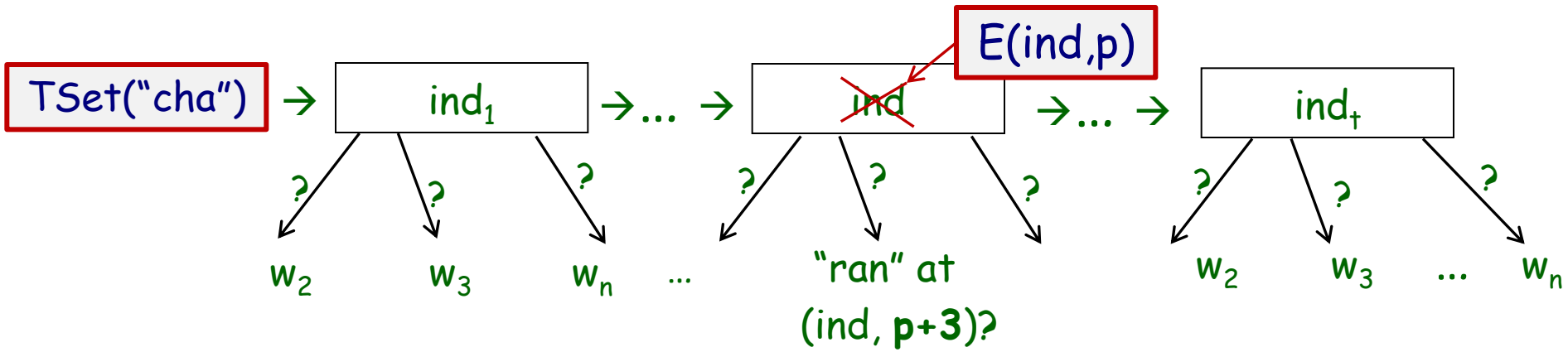
Goal: match relative pos. (query) to absolute pos. (keyword)

W(**ind**) contains "cha" at pos. **p** and "ran" at pos. **p+3**

- TSet("cha") = { ...., E(ind,p), ... }
- XSet contains .... , H("ran",ind,p+3) , ...

Client parses query *charan* as ("cha" & ("ran",3))

- s-term = "cha" ⇨ Eddie retrieves E(ind,p) from TSet("cha")



Substring Search

Goal: match relative pos. (query) to absolute pos. (keyword)

W(**ind**) contains "cha" at pos. **p** and "ran" at pos. **p+3**

- TSet("cha") = { ...., E(ind,p), ... }
- XSet contains .... , H("ran",ind,p+3) , ...

Client parses query *charan* as ("cha" & ("ran",3))

- s-term = "cha" ⇨ Eddie retrieves E(ind,p) from TSet("cha")

We need two-party computation:

Client Input: xtrap("ran",3)

Server (Eddie) Input: E(ind,p)

Eddie's Output: H("ran",ind,p+3)

Substring Search: Conjunction Protocol

Client: strap("cha"), xtrap("ran", Δ) Eddie: $E_{strap("cha")}(ind, p)$

Eddie's Output: H("ran", ind, $p+\Delta$)

Recall Our Regular Conjunctive Protocol (w/o positions):

Client: $PRF_1("cha")$, $PRF_2("ran")$ Eddie: $y = Enc_{PRF_1("cha")}(ind)$

$$H("ran", ind) = (PRF_2("ran"))^{ind}$$

$y = ind / z_{ctr}$, where otp z_{ctr} derived from $PRF_1("cha")$

1. C sends to S : $A = (PRF_2("ran"))^{(z_{ctr})}$
2. S computes $A^y = ((PRF_2("ran"))^{(z_{ctr})})^{(ind/z_{ctr})} = (PRF_2("ran"))^{ind}$

Substring Search: Conjunction Protocol

Client: strap("cha"), xtrap("ran", Δ) Eddie: $E_{strap("cha")}(ind,p)$

Eddie's Output: H("ran", ind, p+Δ)

Modifications to account for positions:

Client: $PRF_1("cha")$, $PRF_2("ran")$, Δ Eddie: $y = Enc_{PRF_1("cha")}(ind)$

$y' = Enc_{PRF_1("cha")}(ind^p)$

"cha" at pos p in record ind

$H("ran", ind) = (PRF_2("ran"))^{ind}$

$y = ind / z_{ctr}$, where z_{ctr} derived from $PRF_1("cha")$

1. C sends to S : $A = (PRF_2("ran"))^{(z_{ctr})}$
2. S computes $A^y = ((PRF_2("ran"))^{(z_{ctr})})^{(ind/z_{ctr})} = (PRF_2("ran"))^{ind}$

Substring Search: Conjunction Protocol

Client: strap("cha"), xtrap("ran", Δ) Eddie: $E_{strap("cha")}(ind,p)$

Eddie's Output: H("ran", ind, p+Δ)

Modifications to account for positions:

Client: $PRF_1("cha")$, $PRF_2("ran")$, Δ Eddie: $y = Enc_{PRF_1("cha")}(ind)$

$y' = Enc_{PRF_1("cha")}(ind^p)$

$H("ran", ind, p) = (PRF_2("ran"))^{(ind^p)}$

$y = ind / z_{ctr}$, where z_{ctr} derived from $PRF_1("cha")$

H PRF under q-DDH for $q = max\ p$

1. C sends to S : $A = (PRF_2("ran"))^{(z_{ctr})}$
2. S computes $A^y = ((PRF_2("ran"))^{(z_{ctr})})^{(ind/z_{ctr})} = (PRF_2("ran"))^{ind}$

q-DDH: given $g, g^x, g^{x^2}, g^{x^3}, \ldots, g^{x^q}$ cannot tell $g^{x^{q+1}}$ from $

Substring Search: Conjunction Protocol

Client: strap("cha"), xtrap("ran", Δ) Eddie: $E_{strap("cha")}(ind,p)$

Eddie's Output: H("ran", ind, p+Δ)

Modifications to account for positions:

Client: $PRF_1("cha")$, $PRF_2("ran")$, Δ Eddie: $y = Enc_{PRF_1("cha")}(ind)$

$y' = Enc_{PRF_1("cha")}(ind^p)$

$H("ran", ind, p) = (PRF_2("ran"))^{(ind^p)}$

$y = ind / z_{ctr}$, where z_{ctr} derived from $PRF_1("cha")$

$y' = ind^p / v_{ctr}$, where v_{ctr} derived from $PRF_1("cha")$

1. C sends to S : $A = (PRF_2("ran"))^{(z_{ctr})}$
2. S computes $A^y = ((PRF_2("ran"))^{(z_{ctr})})^{(ind/z_{ctr})} = (PRF_2("ran"))^{ind}$

Substring Search: Conjunction Protocol

Client: strap("cha"), xtrap("ran", Δ) Eddie: $E_{strap("cha")}(ind,p)$

Eddie's Output: H("ran", ind, p+Δ)

Modifications to account for positions:

Client: $PRF_1("cha")$, $PRF_2("ran")$, Δ Eddie: $y = Enc_{PRF_1("cha")}(ind)$

$y' = Enc_{PRF_1("cha")}(ind^p)$

$H("ran", ind, p) = (PRF_2("ran"))^{(ind^p)}$

$y = ind / z_{ctr}$, where z_{ctr} derived from $PRF_1("cha")$

$y' = ind^p / v_{ctr}$, where v_{ctr} derived from $PRF_1("cha")$

1. C sends to S : $A = (PRF_2("ran"))^{((z_{ctr})^{\Delta} \cdot v_{ctr})}$; and Δ
2. S computes $A^y = ((PRF_2("ran"))^{(z_{ctr})})^{(ind/z_{ctr})} = (PRF_2("ran"))^{ind}$

Substring Search: Conjunction Protocol

Client: strap("cha"), xtrap("ran", Δ) Eddie: $E_{strap("cha")}(ind,p)$

Eddie's Output: H("ran", ind, p+Δ)

Modifications to account for positions:

Client: $PRF_1("cha")$, $PRF_2("ran")$, Δ Eddie: $y = Enc_{PRF_1("cha")}(ind)$

$y' = Enc_{PRF_1("cha")}(ind^p)$

$H("ran", ind, p) = (PRF_2("ran"))^{(ind^p)}$

$y = ind / z_{ctr}$, where z_{ctr} derived from $PRF_1("cha")$

$y' = ind^p / v_{ctr}$, where v_{ctr} derived from $PRF_1("cha")$

1. C sends to S : $A = (PRF_2("ran"))^{((z_{ctr})^{\Delta} \cdot v_{ctr})}$; and Δ
2. S computes $A^{y^{\Delta} \cdot y'} = ((PRF_2("ran"))^{((z_{ctr})^{\Delta} \cdot v_{ctr})})^{(ind/z_{ctr})^{\Delta} \cdot (ind^p/v_{ctr})}$

$= (PRF_2("ran"))^{ind^{p+\Delta}} = H("ran", ind, p+\Delta)$

Extensions

- Wildcards: immediate application of above technique:

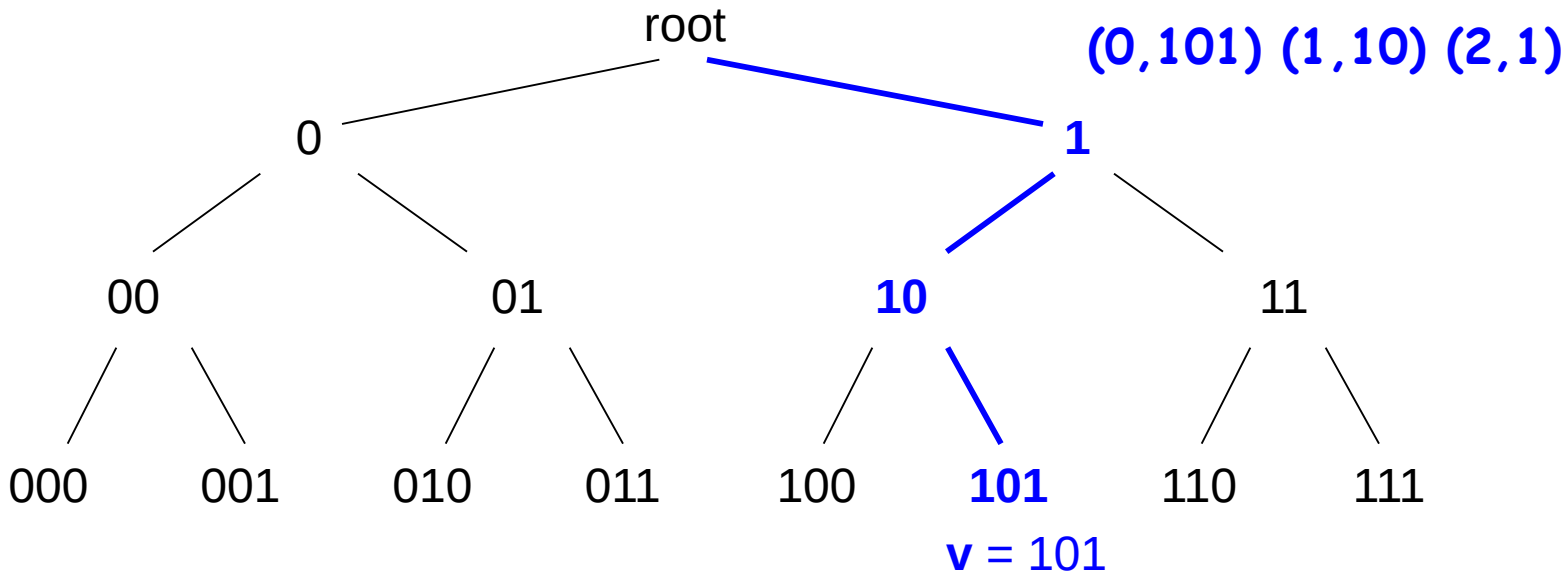
  cha _ _ _ jit: same as (s-term = cha, x-term = jit, Δ =6)

- The described solution assumes s-term is a k-gram, but how about
  - "lname = Jutla" and "name like %ara _ jit" ?
  - We add a data structure XTset which encodes *all* positions of a given k-gram in a record
- Can mix grams of different sizes, e.g. 3-grams as s-terms with 1-grams as x-terms for more flexibility
  - no pre-processing/EDB cost, moderate online overhead (more conjuncts)
- Proximity queries: Phrase queries "Bar Ilan University"

Range Queries

Preprocessing DB

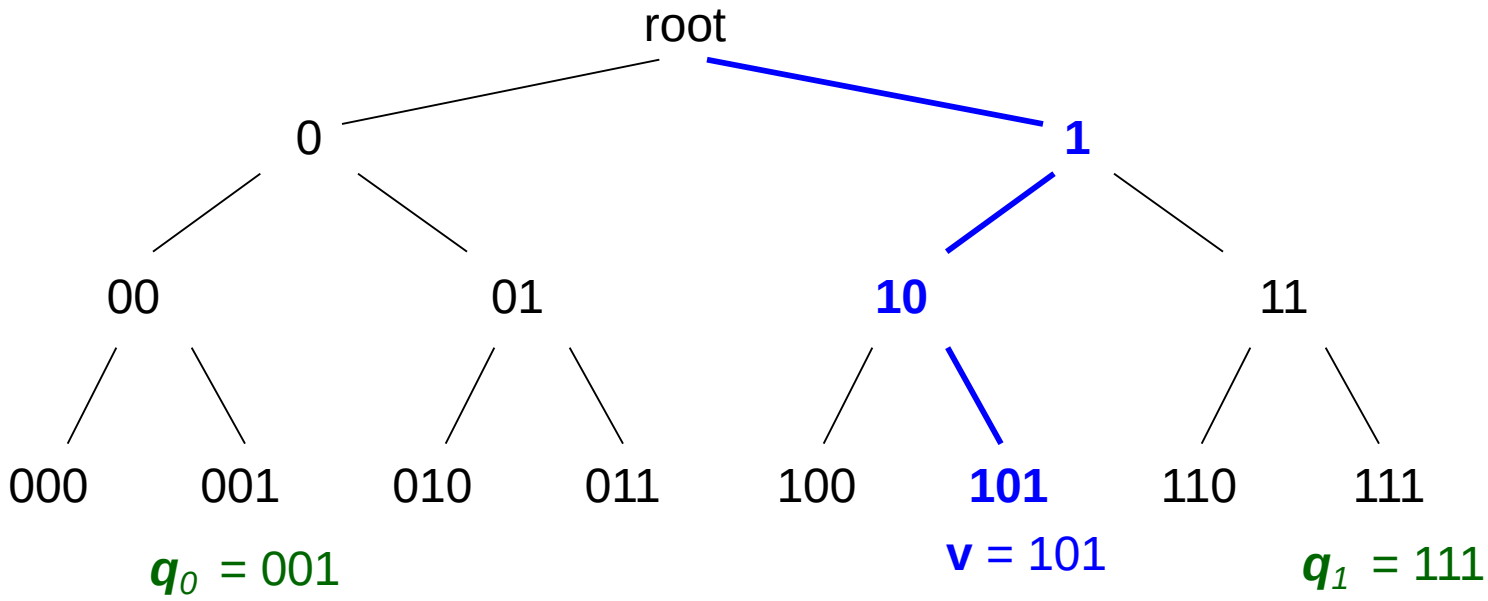
- Attribute valued 0..K=2^k (or any other range)
- Build binary tree with values as leaves (tree height = k = log max range)
- Add k columns: i-th column describes nodes of height i
  - Each new column acts as a new attribute in DB
  - Attribute-value pairs: (height,node)
- Record w/ value v → columns include nodes from v to the root



Query

- Query $[q_0,q_1]$: Client chooses *cover* of $[q_0,q_1]$ interval, namely: $(h_1,c_1),\ldots,(h_t,c_t)$ (c_i describes a node, h_i describes its height)

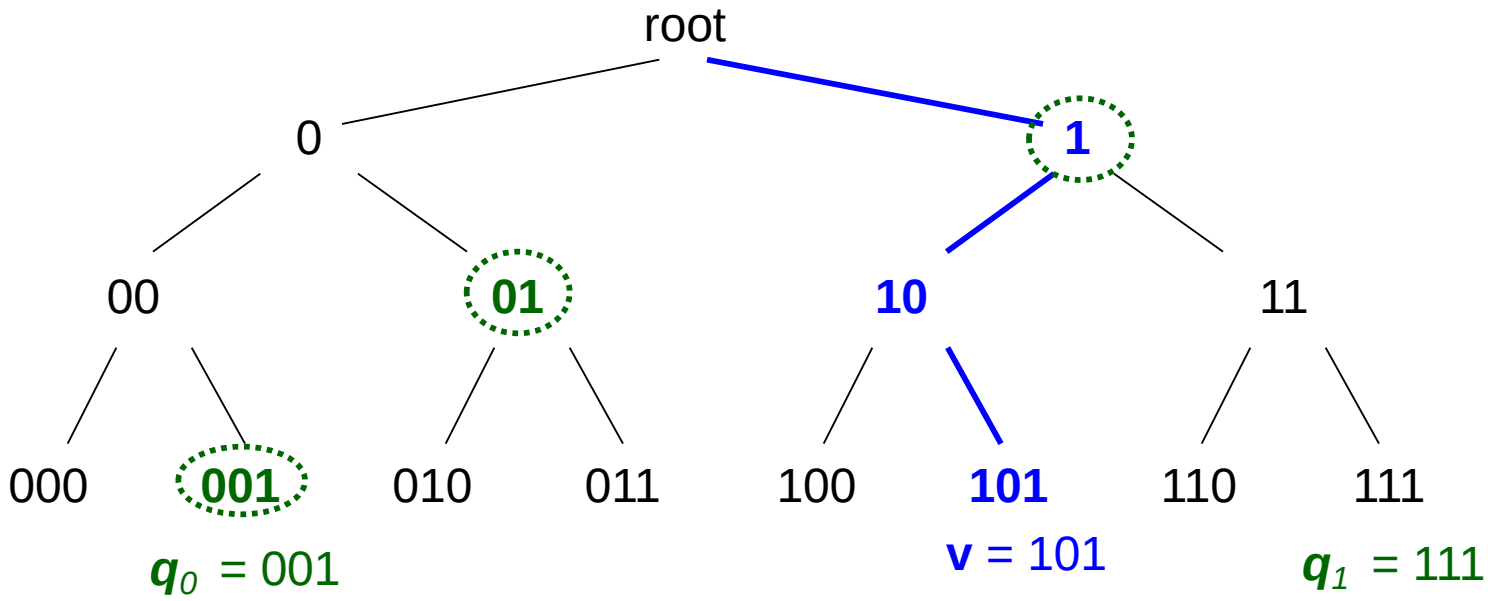
- Client queries a regular disjunction
  - "exact-match(h_1,c_1)" OR … OR "exact-match(h_t,c_t)"



Query

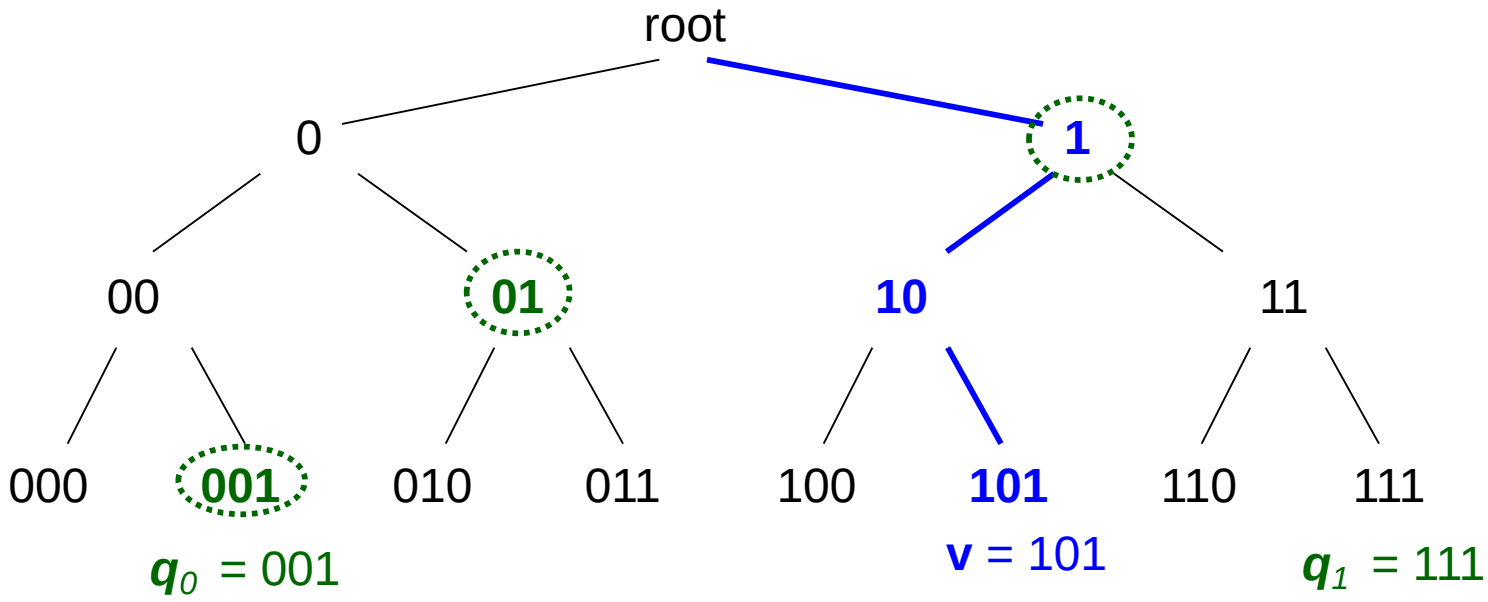
- Query $[q_0,q_1]$: Client chooses *cover* of $[q_0,q_1]$ interval, namely: $(h_1,c_1),\ldots,(h_t,c_t)$ (c_i describes a node, h_i describes its height)

- Client queries "exact-match(h_1,c_1) OR … OR exact-match(h_t,c_t)"
  - e.g. (0,001) OR (1,01) OR (2,1)



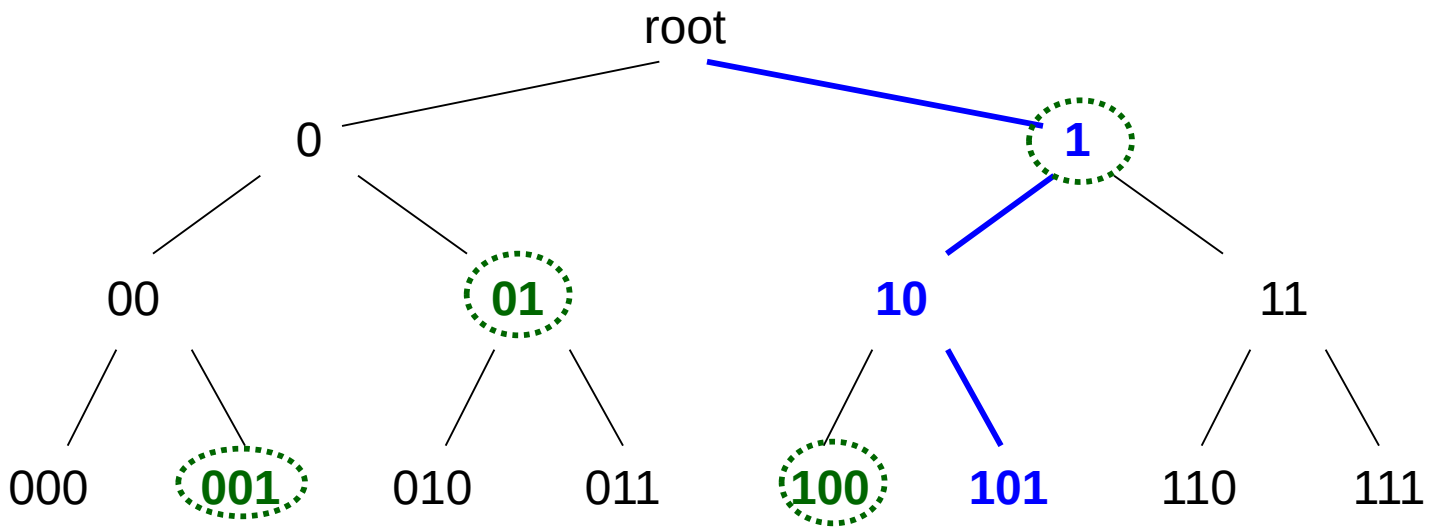
Range Query Authorization

- Our policy authorizes range query based on total size of range
- Client discloses heights to Debbie (the attributes) with which Debbie computes total size (e.g. $2^0+2^1+2^2=7$)
- Client is allocated a max allowed range by policy
  - we do not guarantee contiguous range



Privacy Concern

- Assume client always chooses a minimal cover (min # nodes)
- Client discloses heights (the query attributes) to Debbie
- Debbie learns total size (good) but…
- … can distinguish b/w different ranges of a given size (bad)
- E.g. [4,7] has cover w/single node while [1,4] needs 3 nodes



Privacy-Preserving Covers

Universal Covers

- Def: The *profile* of a cover is the set of heights: eg. {0, 0, 1} vs {2}
- Are there *universal covers*? I.e. a way to choose covers such that all ranges of a given size have the same profile?
- Answer is yes (e.g. set of leaves). More interesting (*minimal universal*)
  - Size 20 = 15+5 → (1+2+4+8) + (1+4) → profile (0,1,2,3,0,2)
- We use universal covers to hide anything but total size from Debbie
  - We call these "canonical covers"

3-node universal covers

- Canonical covers: up to 2 log n nodes – can we have them smaller ?
- Not possible in general, except if we're willing to expand the range
- 3-node universal *over-covers* exist for all ranges (40% avg overhead)
- Offers tradeoffs in performance and leakage

Leakage from Range Queries

For Debbie:

- The total size of the queried range (necessary to apply policy)

For Eddie:

- Leakage for atomic range query with cover $w_1=(h_1,c_1)$ … , $w_n=(h_n,c_n)$ is same as for OXT disjunction "w_1 or … or w_n"
  - $|DB(w_i)|$ for i=1,…,n , $DB(w_1$ or … or $w_n)$ (3-node solution better here)
- For composite queries, leakage is same as OXT where the range query is replaced with "w_1 or … or w_n"

For Client:

- Mask($|DB(w_i)|$) for i=1,…,n if range query acts as s-term

Time/Space Overhead for Substring and Range Queries

- Both are non-interactive as any other queries in OXT (one msg from C, matching encrypted ind's from E)
- Substring/wildcards queries
  - Space: ~1.8 times tuple size (Ph 1), O(n) tuples for each n length field
  - Online: n/4 exponentiations for n-long substring/wildcard query
    - "4" is from 4-grams
- Range queries
  - log N new columns per range-searchable attribute
    - N=max searchable range size
  - ~N Tset's
  - Online: (log n)-term disjunction (n = size of *queried* range), or 3 in 3-node

Subsequence Generalization: Proximity Queries

- A generalization of our substring technique
- Can do search of the form (e_1, e_2, Δ) meaning
  - Return all records where element e_1 is at distance Δ from e_2 (Δ can be negative)
- Examples:
  - e_i are k-grams: resolves substrings and wildcards
  - e_i are textual words: resolves phrases (e.g., "Bar Ilan University")
  - Multi-dimensional distances (e.g., grid), etc.

Subsequences Leakage to Eddie

Reminder: Leakage from conjunctions $w_1 \wedge \ldots \wedge w_n$

1. Index size = upper bound on Σ_i $|Doc(w_i)|$
2. Number of terms in each conjunction
3. Size of s-term set $|Rec(w_1)|$ and whether s-term repeats
4. Size of $Rec(w_1 \wedge w_j)$, $j=2,\ldots, n$
5. For queries $w_1 \wedge x$ and $w_1' \wedge x'$, if $x=x'$ and $Rec(w_1 \wedge w_1') \neq \phi$, then E learns that $x=x'$ and the encrypted rind's in $Rec(w_1 \wedge w_1')$

Leakage is similar for subsequence queries with s-term k-gram w_1 and x-term grams $w_2,..,w_n$ except for more involved 5' (stated for 2-term query)

- For queries $w_1 \wedge x$ and $w_1' \wedge x'$, with offsets Δ_1, Δ_2 , if $x=x'$ and there exist ind in $Rec(w_1 \wedge w_1')$ and p, p' such that w_1 is in position p in ind and w_1' in position p' in ind, and $p-p'= \Delta_1- \Delta_2$ then E learns that $x=x'$, and the encrypted pairs (ind,p), (ind,p').
- Plus: offset Δ leaks – can be avoided with a round of communication C-E