Blind Seer:
Scalable Private DB Querying

Columbia-Bell Labs work on IARPA SPAR project

Vladimir Kolesnikov (Bell Labs),

Steve Bellovin, Seung Geol Choi, Ben Fisch, Angelos Keromytis, Fernando Krell,

Tal Malkin, Vasilis Pappas and Binh Vo (Columbia)

Wesley George (UToronto), Abishek Kumarasubramanian (UCLA)

Applied MPC Workshop Aarhus

Outline

- Project
- Basic system architecture
- Basic approach
- Additional features
  - Protection against malicious players
  - Other interesting issues/solutions

IARPA SPAR Program

- Tim Edgar mentioned the origins of IARPA SPAR
- Result of ODNI asking itself the question of privacy?
  - There exists Deputy for Civil Liberties for the Director of National Intelligence

This Project

- Solves specific problem
- Improves/clarifies state of the art of large GC-based SFE.

- Basic approach to appear in Oakland 2014
- Pappas, Krell, Vo, Kolesnikov, Malkin, George, Keromytis, Bellovin, “Blind Seer: A Scalable Private DBMS”

Required features

100M records, 10TB DB
Preserve query and data privacy

Allowed up to 2-10x overhead compared to MySQL

Robust query support:

```
select * where NAME=Bob AND AGE >20
```

Boolean query expressions (including at least three conjunctions)
Range queries and inequalities for integer numeric, date/time, etc
Matching of keywords —close to a specified value (stemming)
Text fields with many keywords (e.g. 100's)

Matching of values with wildcards
Matching of values with a specified subsequence
m-of-n conjunctions
Ranking of results
…

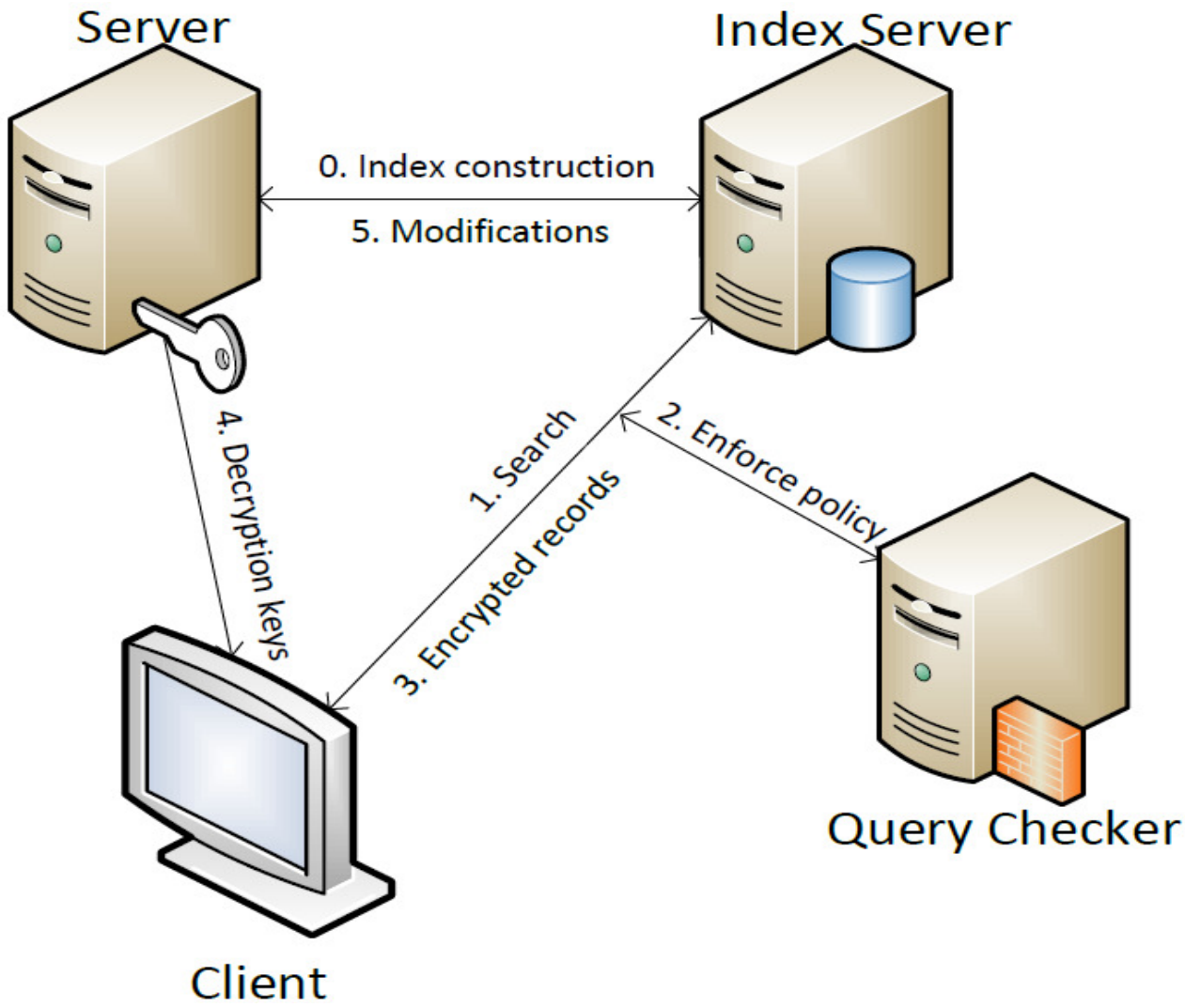
Blind Seer: The Basics

- Requirements are very hard to achieve securely (impossible…?)
- must relax security guarantees
- Challenge: How? Find a right TRADEOFF
  - meaningful, reasonable, provable security (controlled leakage)
  - Lots of interesting research to be done…

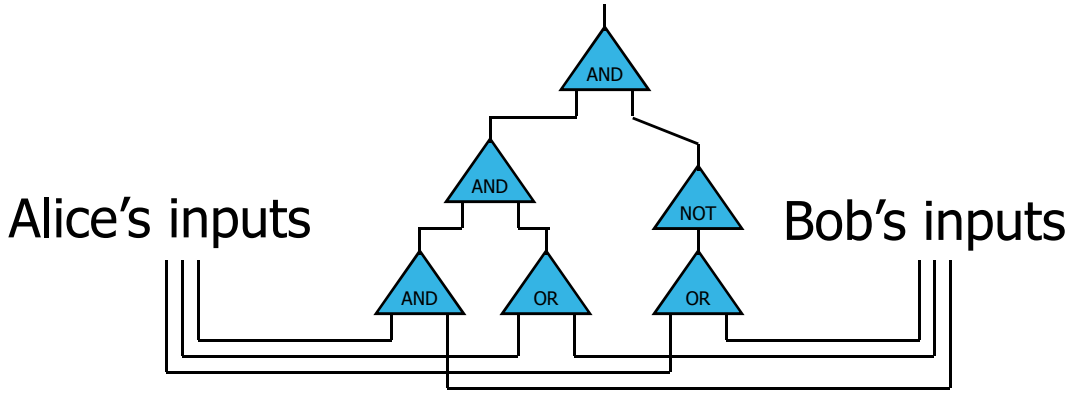
    How much leakage is too much? How do you evaluate the damage of a certain leakage profile? How do you qualify/quantify how good your tradeoff is? Application specific? Is there a formal way to do so?

- I will describe Our basic system and approach:
  - Touch upon some of our tradeoffs
  - Focus on Boolean queries, semi-honest parties
  - Briefly discuss migration to Malicious

System Architecture



Secure Computation: Yao's GC



Very fast for small problems, but doesn't scale to large circuits

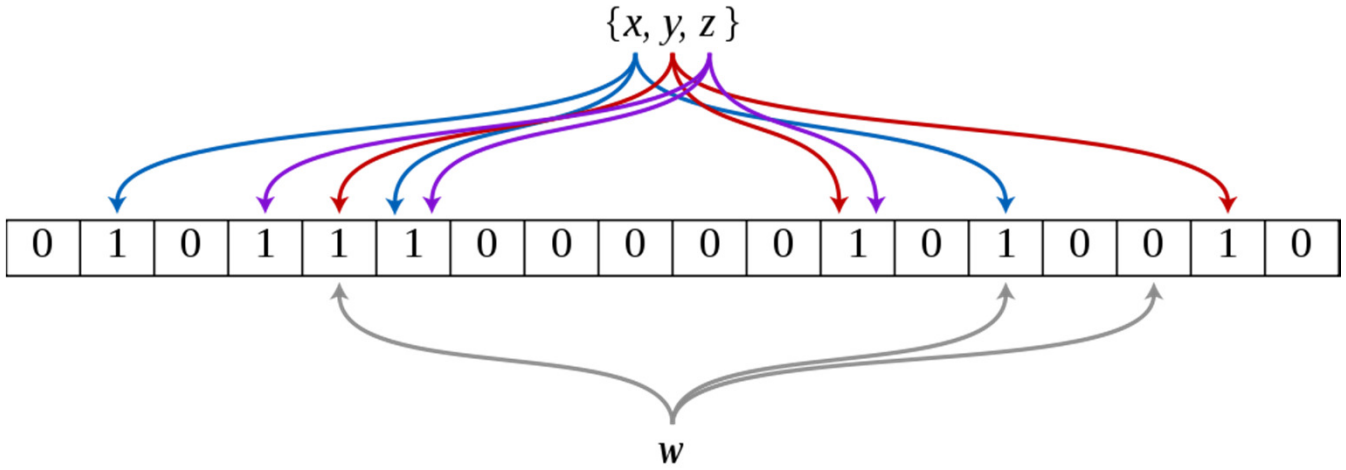
How to scale?

- Identify privacy-critical subroutines and implement securely
- Insecure implementation of the rest

Challenge: Understand and formalize security guarantees (hard problem)

Bloom Filter

Constant-time querying
Efficient storage (ca 10 bits per keyword)
Fixed access pattern (same for both match and non-match)

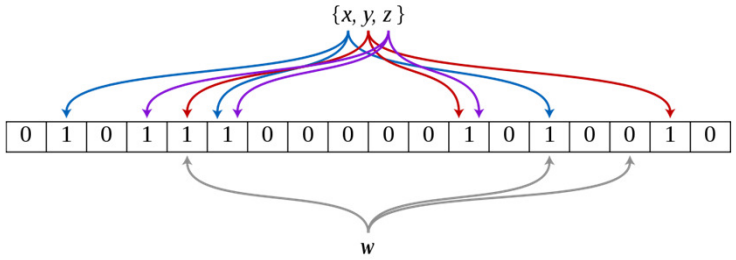


Encrypted BF:

- **Same as BF, but objects are encrypted – need deterministic encryption**

Occluded BF

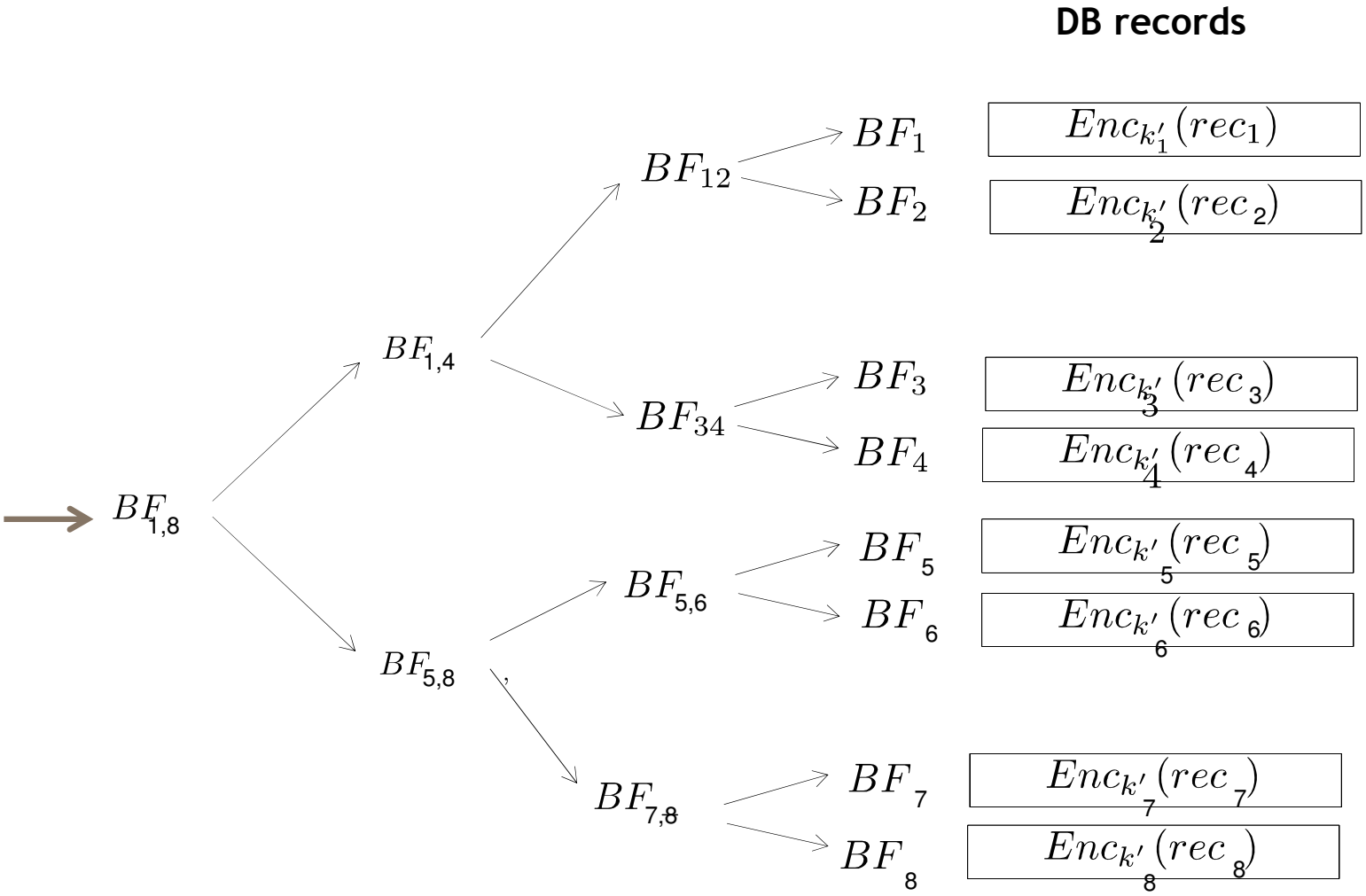
Query: C sends Enc(kw), S computes match
OK for single keyword searches
For formulas, need to hide terms matching



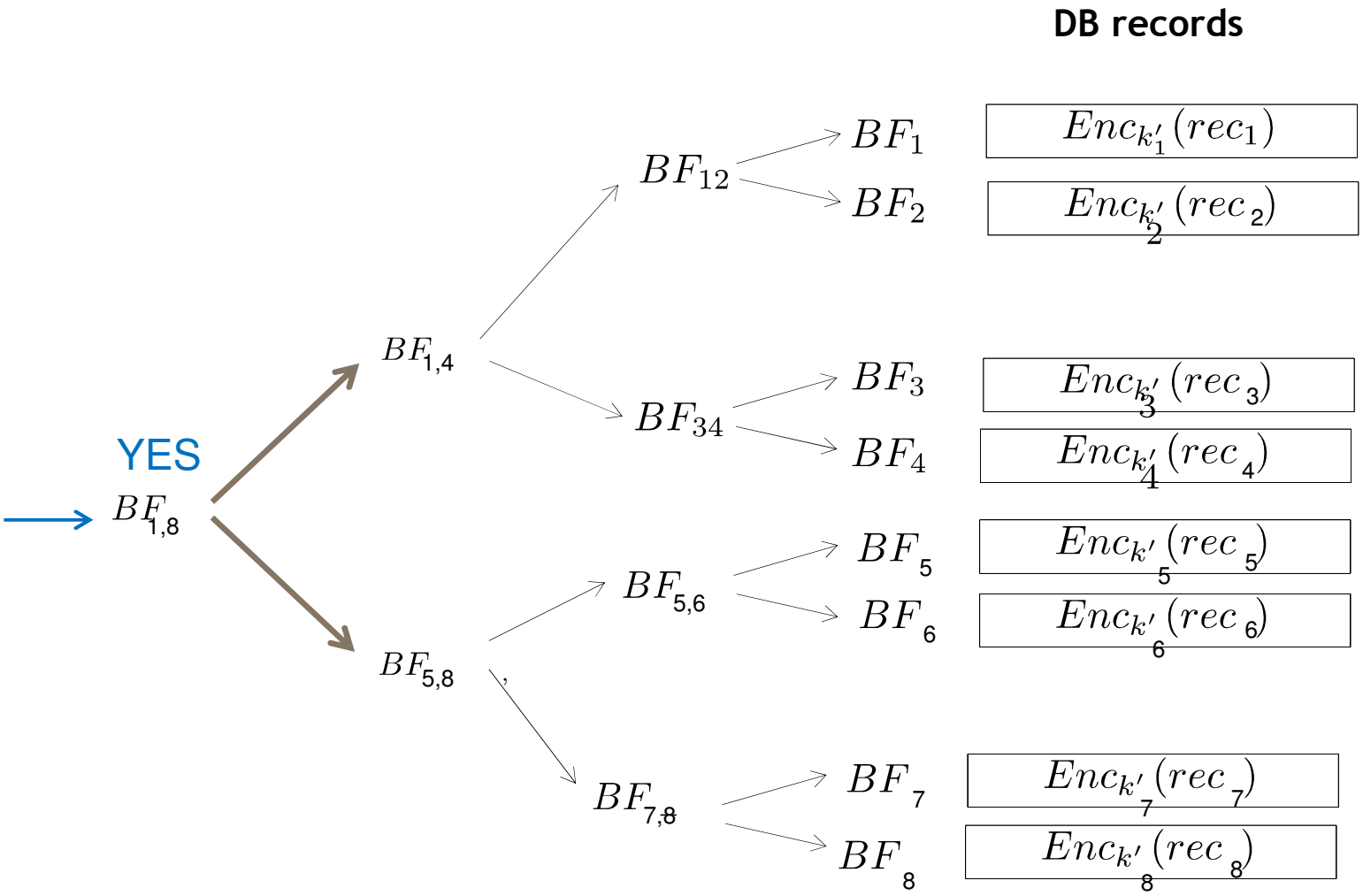
Idea:
Mask BF with a (pseudo-)random pad
Let Client know the pad (via seed)

Then Client and Server run SFE for computing match, where C inputs pad.
GC is very efficient: 10-20 gates per term, plus gates to implement formula.

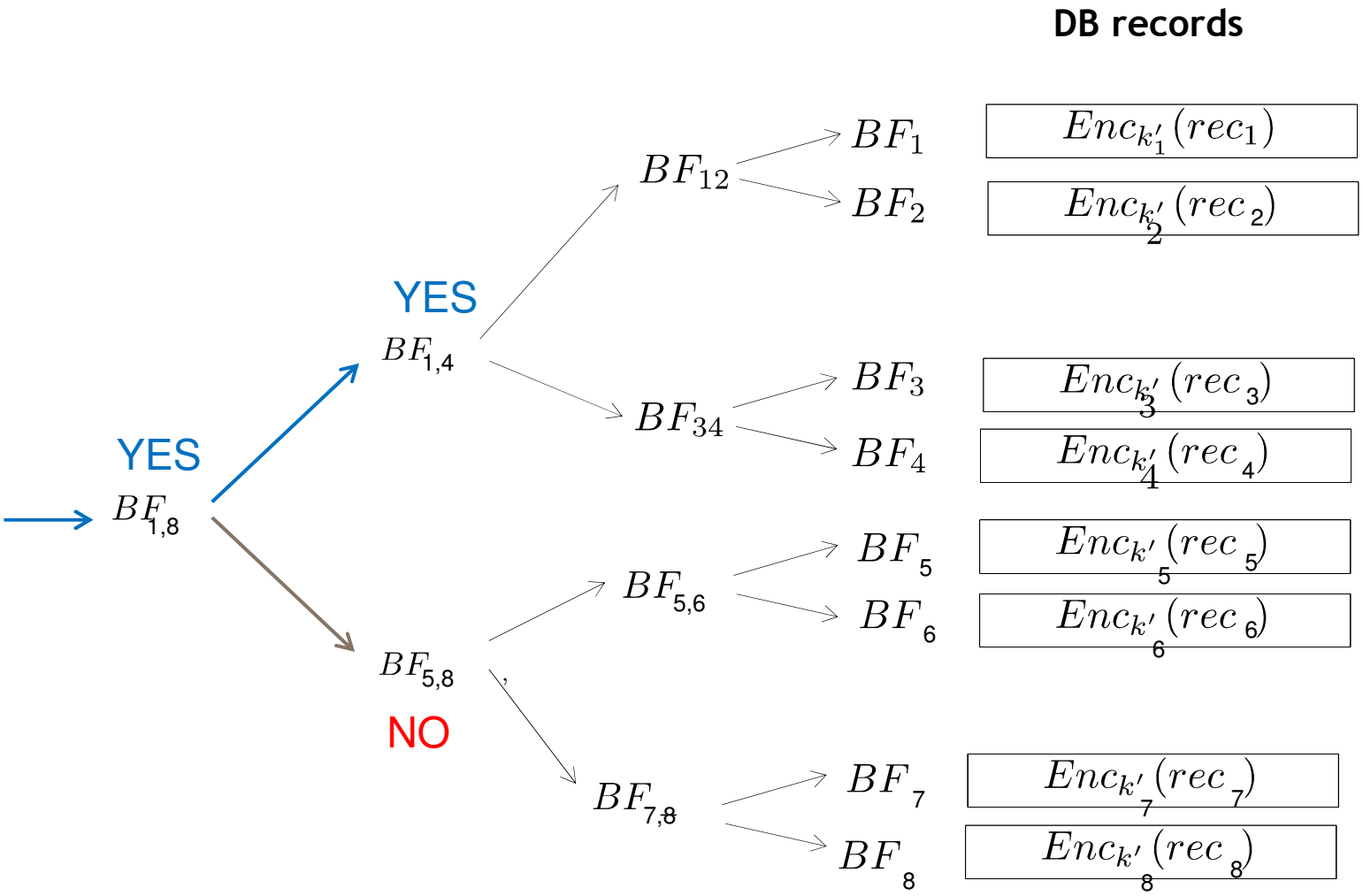
Search Tree


DB records
$BF_{1,8}$
$BF_{1,4}$
$BF_{5,8}$
BF_{12}
BF_{34}
$BF_{5,6}$
$BF_{7,8}$
BF_1
BF_2
BF_3
BF_4
BF_5
BF_6
BF_7
BF_8
$Enc_{k'_1}(rec_1)$
$Enc_{k'_2}(rec_2)$
$Enc_{k'_3}(rec_3)$
$Enc_{k'_4}(rec_4)$
$Enc_{k'_5}(rec_5)$
$Enc_{k'_6}(rec_6)$
$Enc_{k'_7}(rec_7)$
$Enc_{k'_8}(rec_8)$

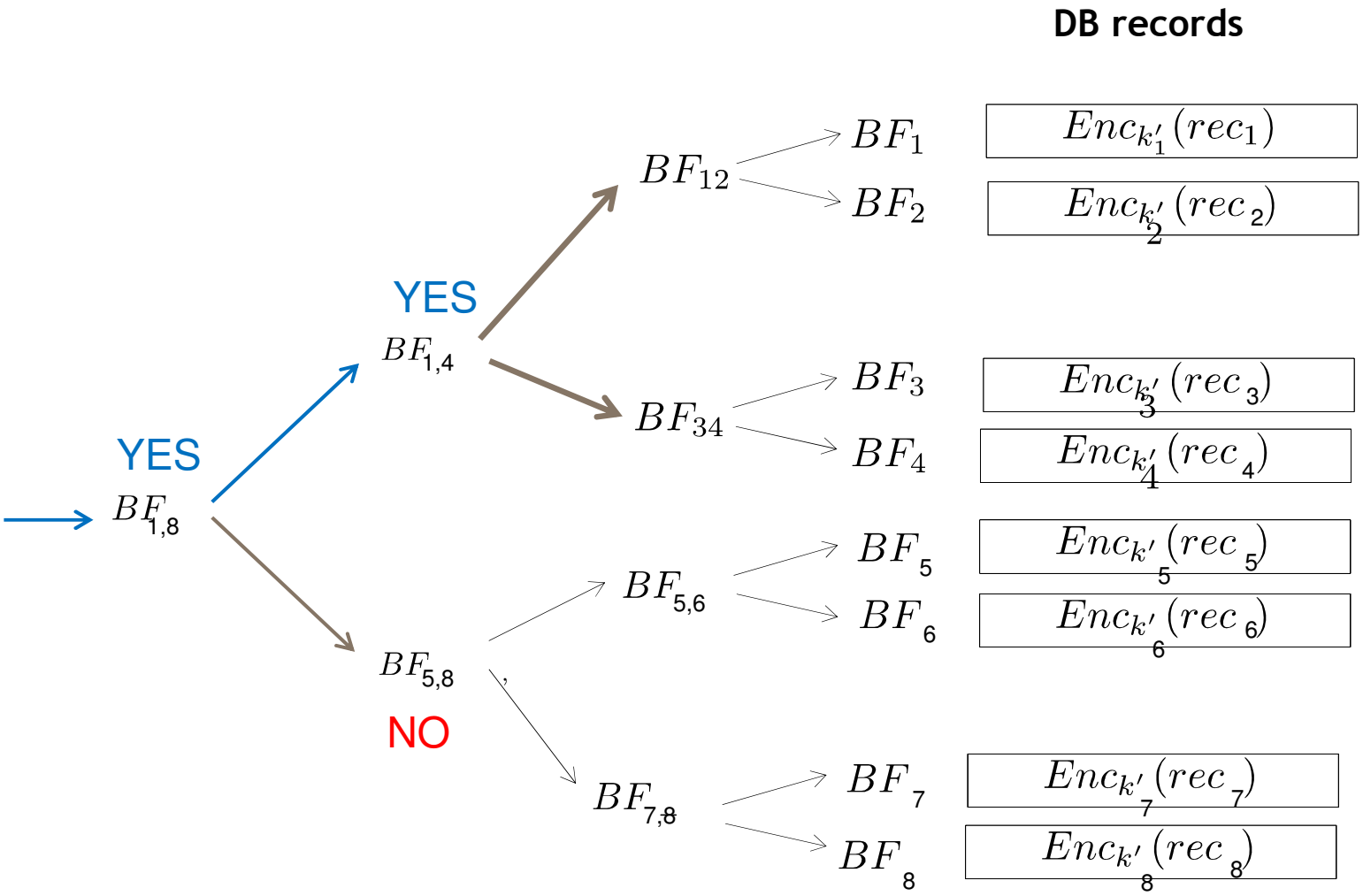
Search Example



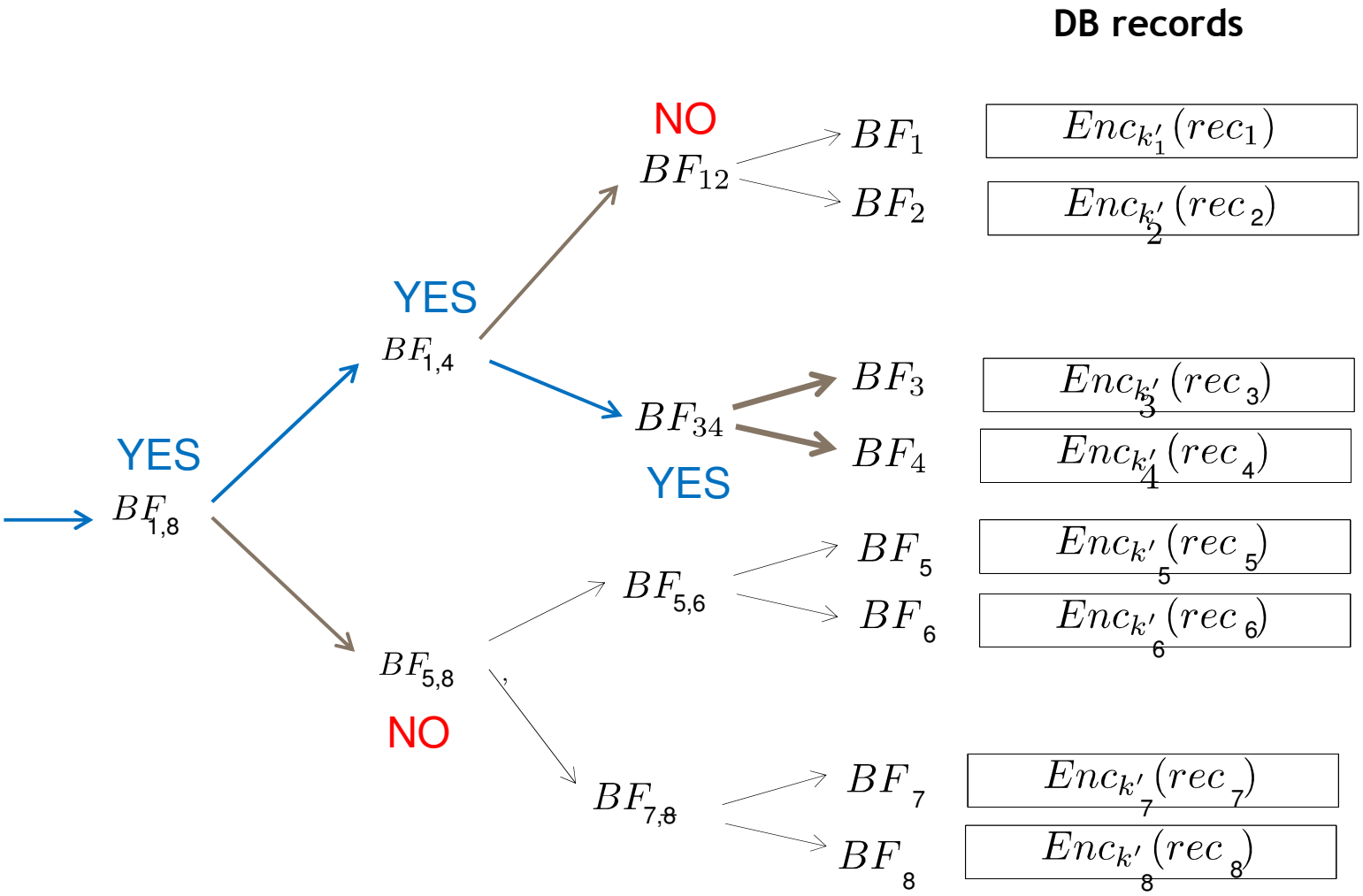
Search Example



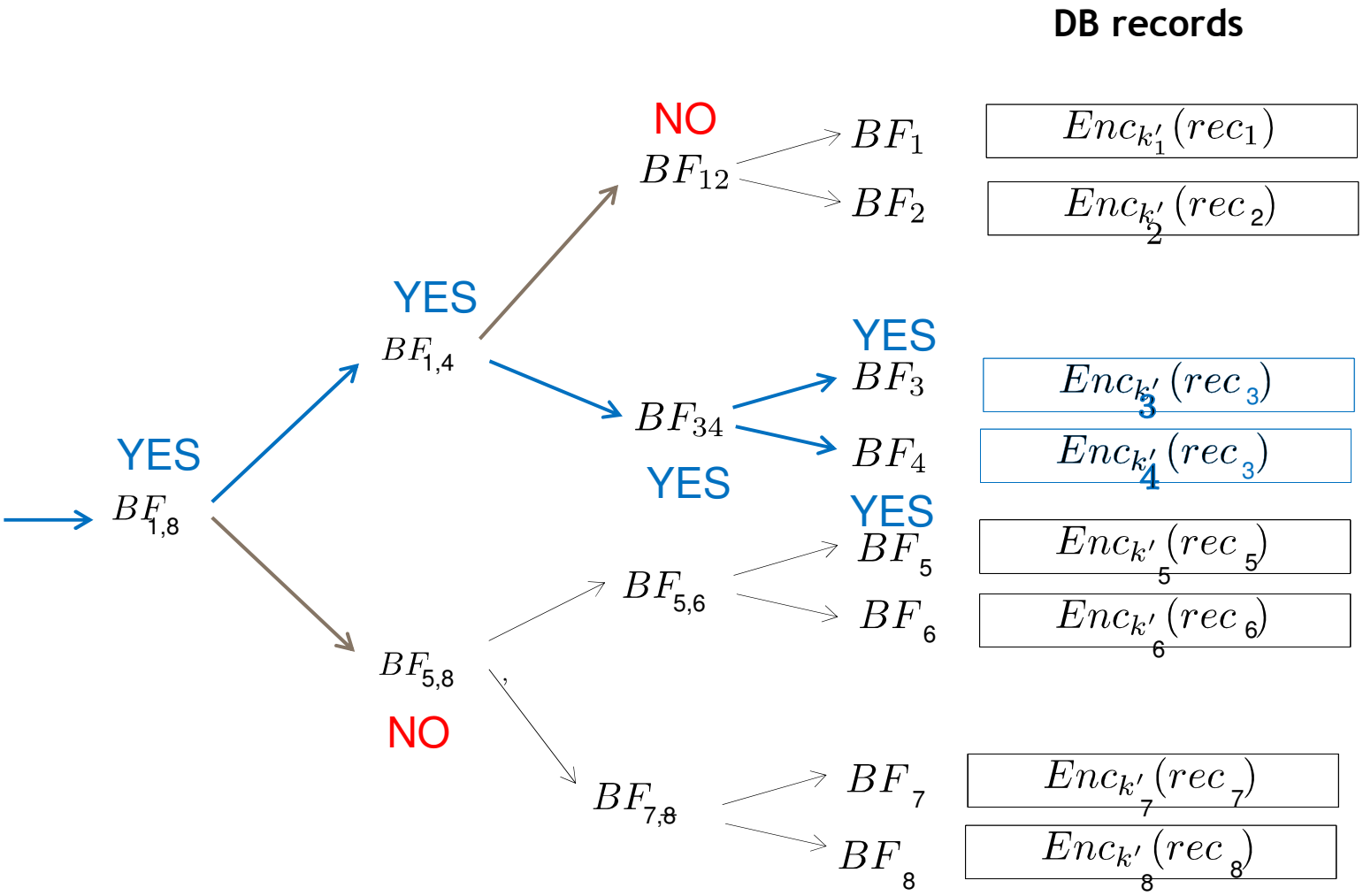
Search Example



Search Example



Search Example

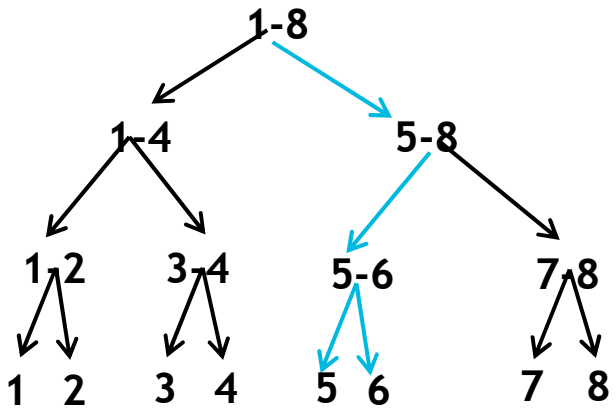


What is Leaked ?

- Query Pattern (e.g., S can distinguish between simple and complex queries, may learn about repeated terms in different queries)
- Returned Records Access Pattern
- Tree search pattern of each query:

What is Leaked ?

- Query Pattern (e.g., S can distinguish between simple and complex queries, may learn about repeated terms in different queries)
- Returned Records Access Pattern
- Tree search pattern of each query:



What does this mean?

OR queries:

- Only leakage is access patterns (tree traversal can be simulated, no leakage on individual terms)
- Efficiency proportional to number of results (asymptotically optimal)

AND queries:

- Tree search pattern reveals more: also abandoned paths
- Efficiency: at most proportional to number of matches for best term (asymptotically optimal??)

  Similar to MySQL, but don't need to know which term is best
- Abandoned paths (leakage, run time) depend on data and query (and randomness of tree construction).

Arbitrary Boolean Formulas

- Efficiency: at most proportional to number of matches of BEST term in CNF decomposition of formula

  (don't need to know it)

- Leakage: access pattern and tree search pattern
  - Hard to quantify but "much less" than giving information on patterns of individual terms in the formula

  (no information on plaintext beyond patterns)

0-1 Result Set Size Indistinguishability

Goal: hide from S whether there was a 0 or 1 match.

S is an airline and C is gov't querying for POI. Expect 0 hits

S learning of a match can cause panic.

Def 1: Consider probability of bad event, prove it's small

Def 2: If distinguishable, guarantee that D's confidence is not very high

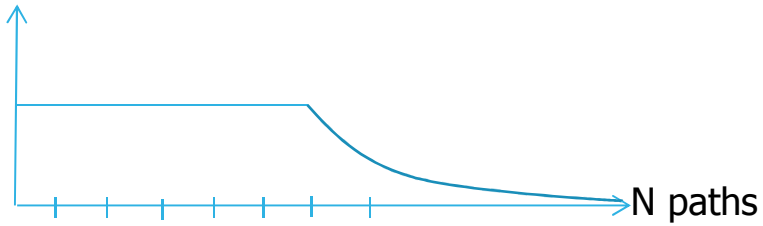
0-1 Result Set Size Indistinguishability

Goal: hide from S whether there was a 0 or 1 match.

Def 2: If distinguishable, guarantee that D's confidence is not very high

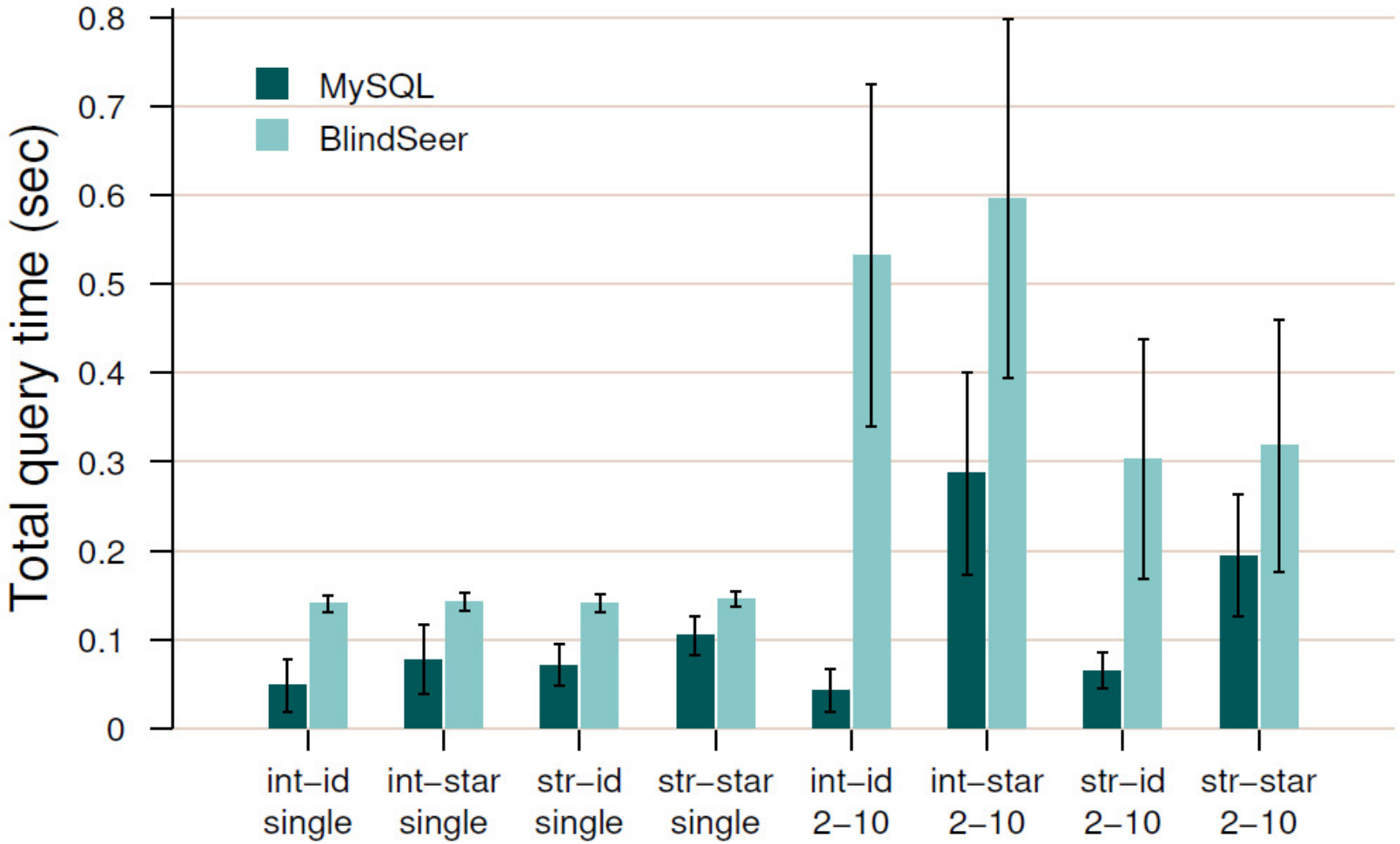
- if the a-priori probability of a 1-case is δ, then conditioned on any possible view, the a-posteriori probability of a 1-case is at most $(1+\epsilon)\delta$).

Solution: C adds p of fake tree-traversal paths. p is a random variable drawn from distribution like this

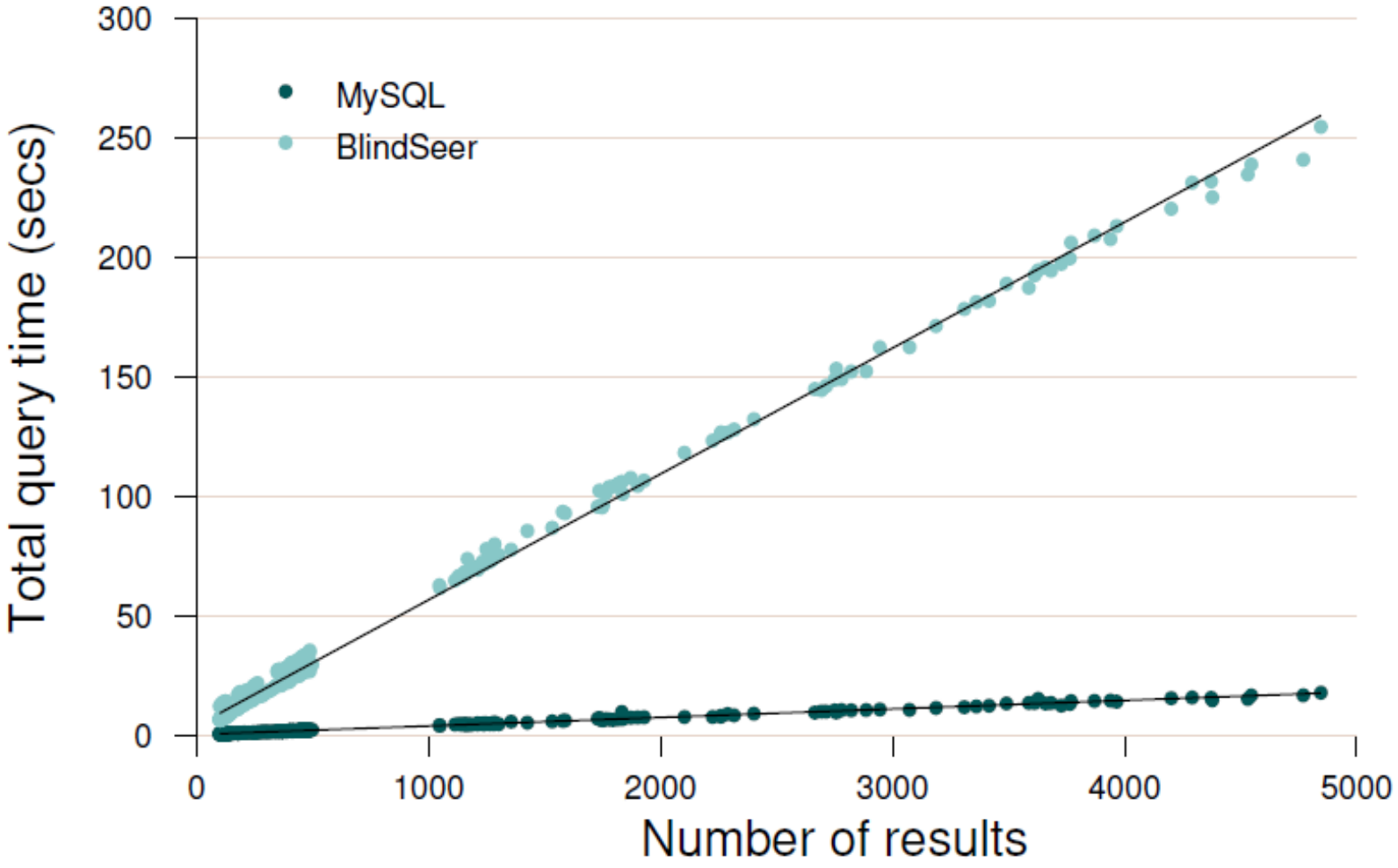


Theorem: Above solution satisfies Def. 2 with ϵ=1

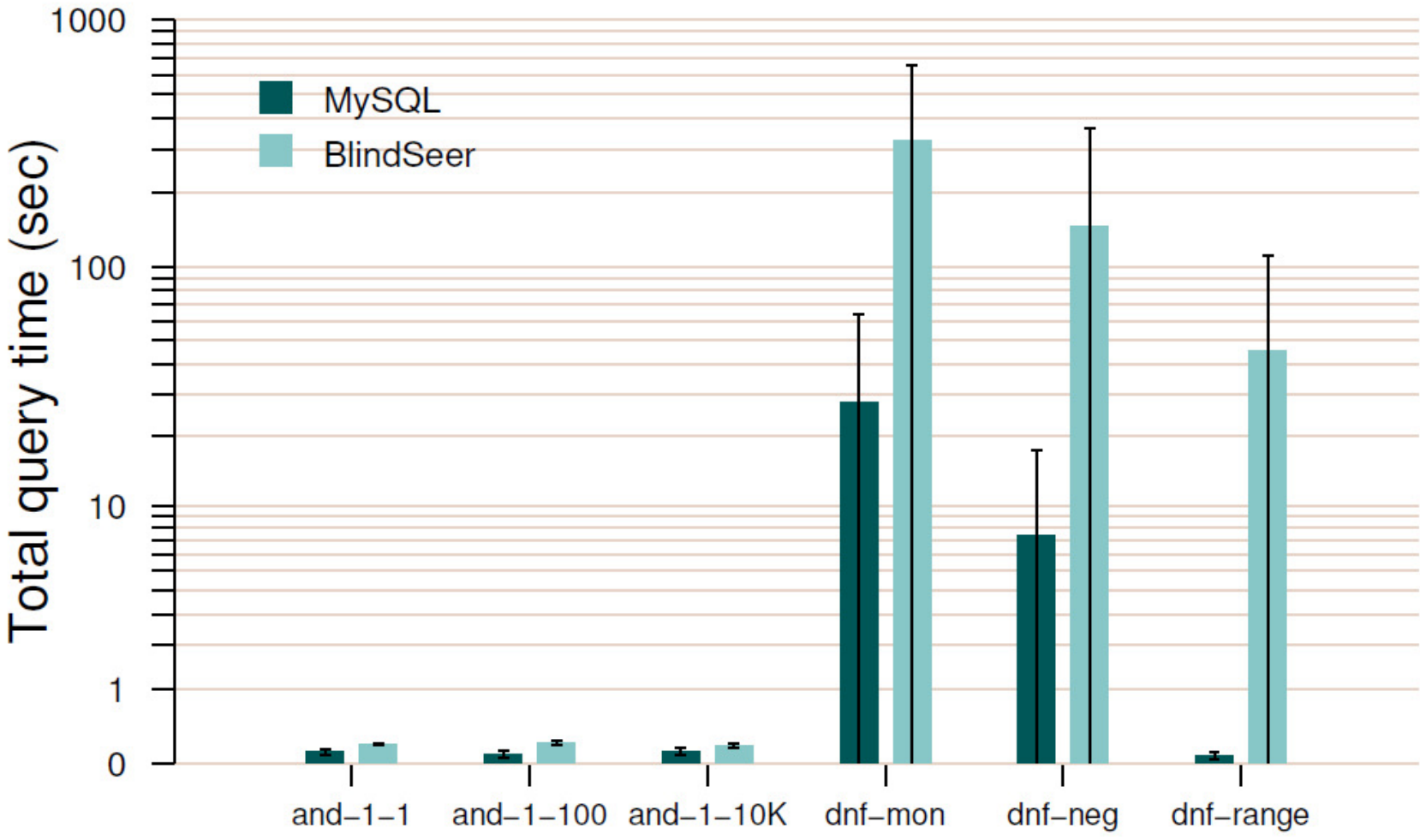
MySQL comparison



Result set scaling



Boolean query performance



Optimizations, Advanced Features, And More

Optimizations

- Parallelization
  - Naïve parallelizing multiple queries
    - Demonstrated up to 5x throughput benefit
  - More intelligent speedup intra-query
    - Tree traversal is the most expensive step
    - Node visits are mostly independent, and thus highly amenable to parallelization
    - In 16-core setup observed 16x improvement for traversal.

Optimizations

- Better BF analysis
  - Currently 10^-6 false positive rate in all tree nodes
    - Much smaller FP rate is sufficient to achieve total 10^-6
    - BF FP are good for security:
      - Creates noise in tree traversal patterns
  - With 10^-3 FP, # circuit inputs is cut in half, and hence expect approx factor 2 performance improvement.
  - Observed about 50% improvement

Optimizations

- Faster secure computation with GESS (vs garbled circuit)
  - Kolesnikov-Kumaresan (SCN 2012)
    - Approx 3x improvement in bandwidth vs best GC
      - Greatly improves performance on narrow channels
    - Working on translating this into speed improvement on fast LAN.
    - Observed about 50% improvement in experiments
    - Did not integrate into current code so far

Optimizations

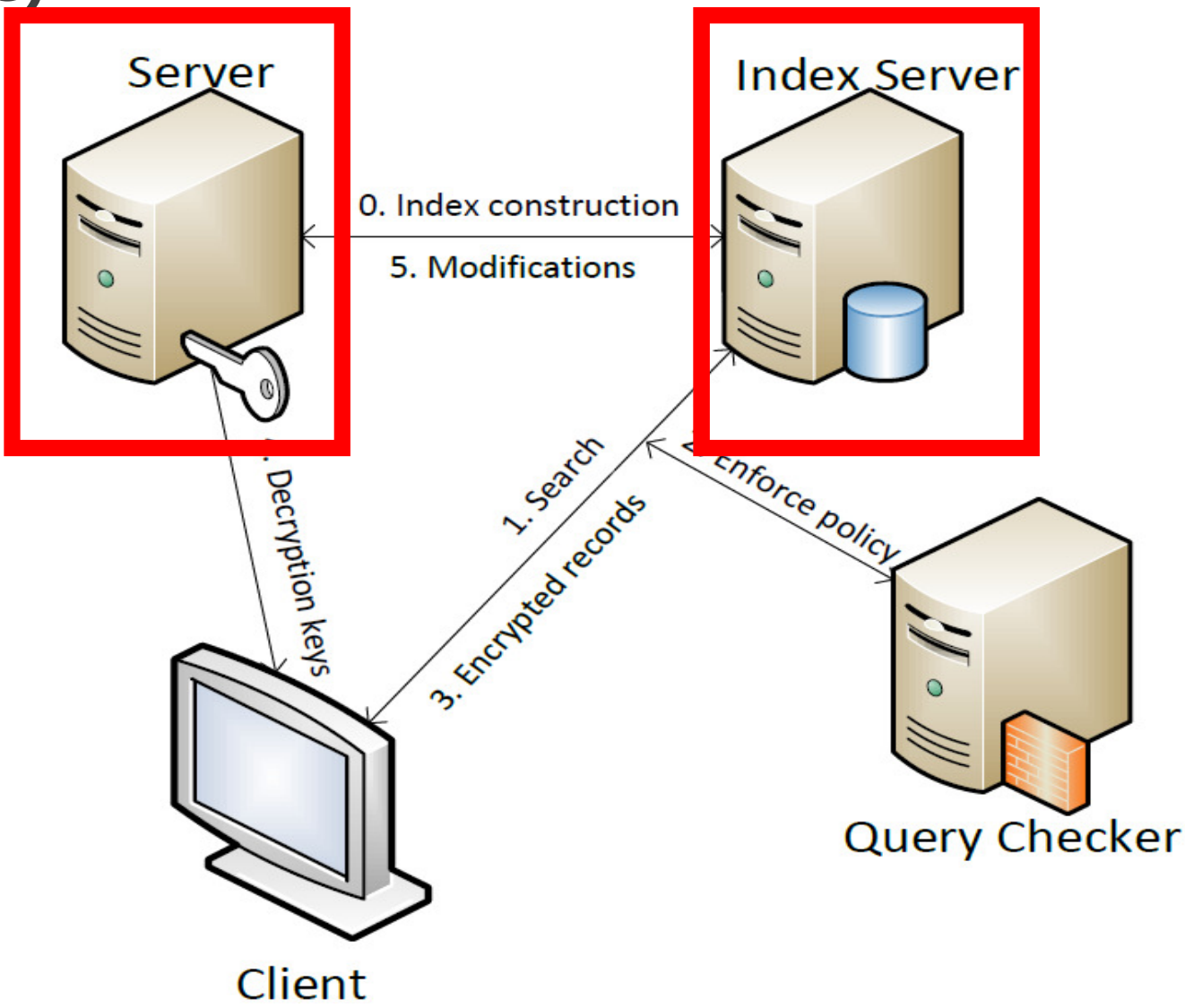
- Code optimization
  - Possibly most important for performance but effort-intensive
  - Interplay of LAN, caching, threads, RAM access

  - Plugging in improved OT extension code of ALSZ13 (CCS 2013) (one of tricks their tricks is also in Kolesnikov-Kumaresan13 (Crypto 2013))
    - Factor 3 improvement in OT => factor 1.5-2 overall (?)

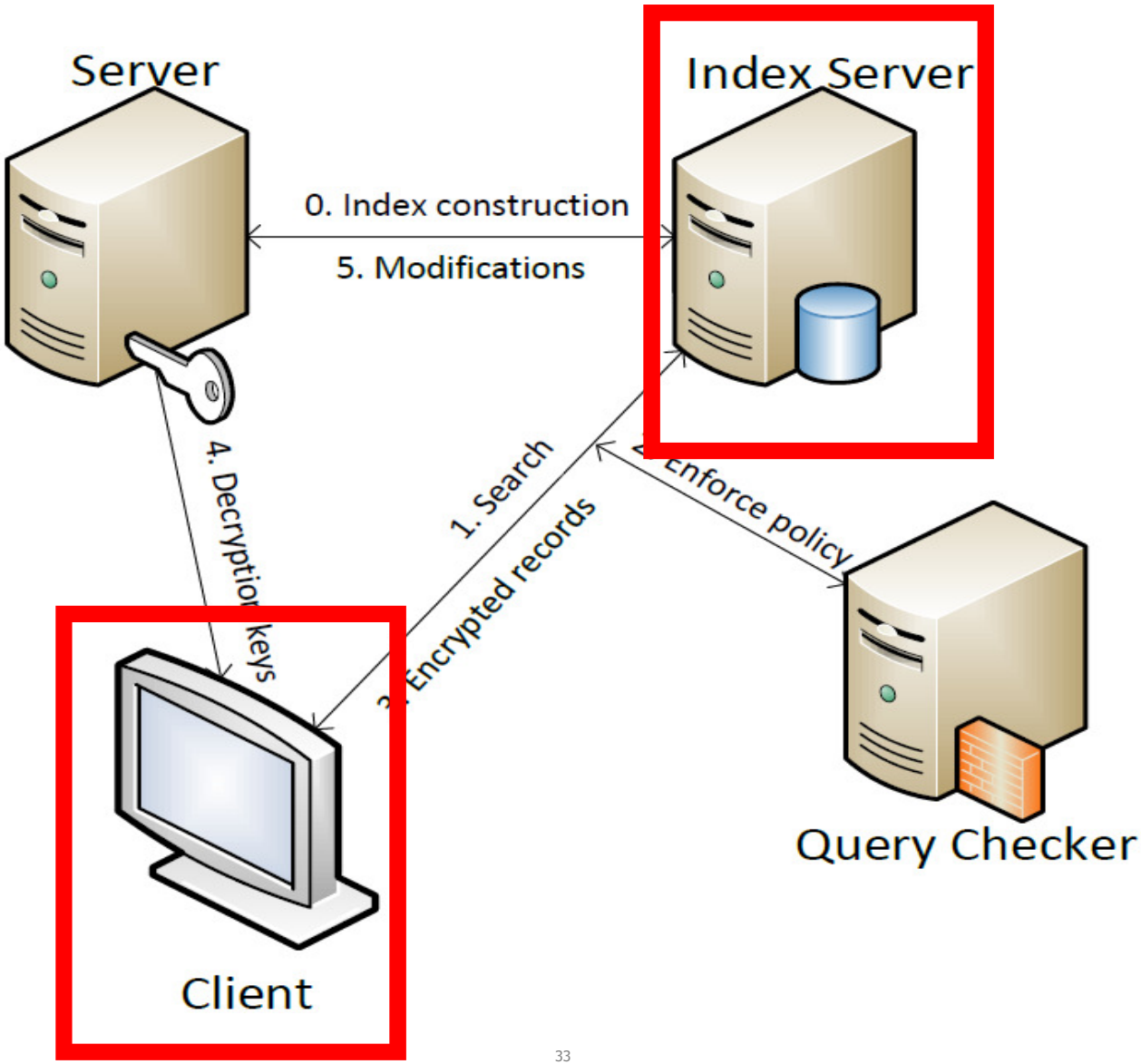
    - Conflicted with the multithreaded libraries we used;
    - Did not integrate in our code

Privacy improvement: Search Tree Rebuilding

Why Rebuilding – Resource utilization (preprocessing phase)



Why Rebuilding – Resource utilization (query phase)



Search Tree Rebuilding

- Use two Index Server boxes and switch between them
- Algorithm
  - Same as preprocessing algorithm with fresh randomness
  - No re encryption of records needed – Really fast
  - RAM-only computation
- Efficiency
  - Only the bloom filter index tree is rebuilt
  - Takes about 20 min to generate and transfer (vs 1-3 days of full DB init)
- Security across rebuilds
  - Tree traversal information learnt by the client/IS is much less useful.
  - IS now sees new BFs with new hash fcns.

Policy Compliance

GC is strategically at the center of our approach because easy to compose.

Requirement: secure policy checking:

Policy rejection should look like a query no-match to C and S

implement policy as a GC computation whose output is an input to BF tree node GC computation.

Malicious Client Protection

- Guarantee (roughly):
  - Actively cheating client cannot receive DB rows if the query is unauthorized.

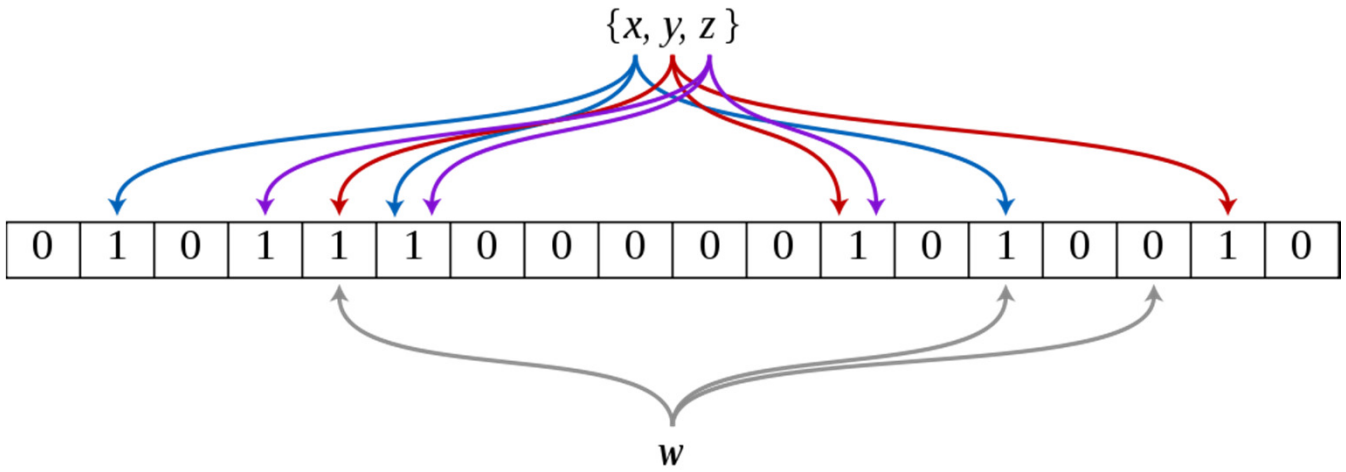
- Idea:
  - GC is secure against malicious evaluator.
  - Have IS generate the garbled circuit

Malicious Client

- Secure policy enforcement and database privacy against maliciously behaving clients who may arbitrarily deviate from protocol
- Why malicious? So far,
  - Query sent to the Query Checker which enforces policy can be completely different from the query sent to the Index Server
  - Client can change his decryption mask at will

Client Changing his mask

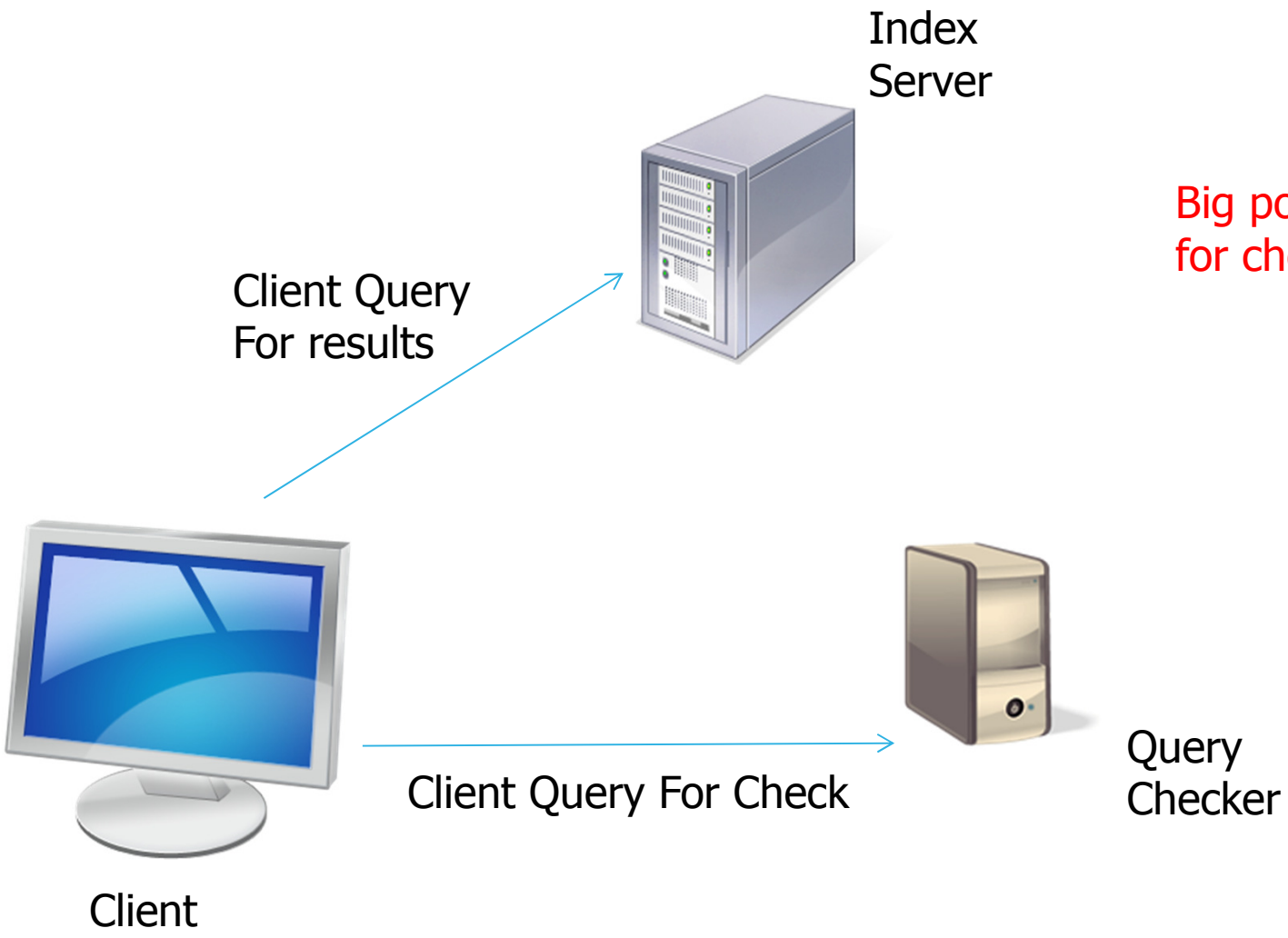
Recall:
We have Encrypted BF
Decryption Key is with the client – What if a Client changes it?



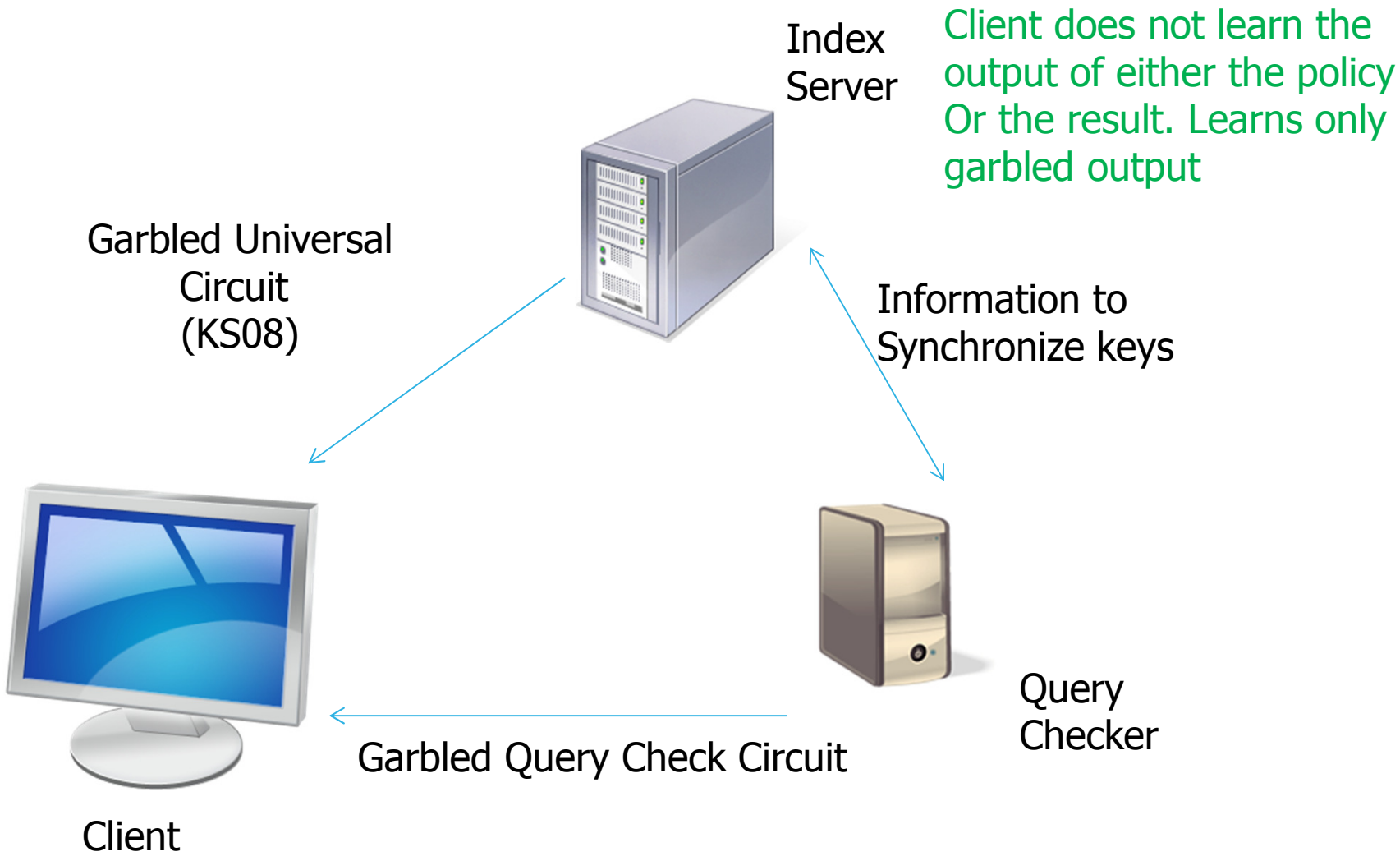
Client changing his mask

- We provide analysis that asymptotically, a client changing his mask is not likely to succeed with sufficiently small probability
- BF uses 30 bits per keyword and 20 hash functions
- The average density of a bloom filter is about 1/2 with the false positive rate of 10^-6. There are tons of zeros when a keyword does not match
- Flipping a random false negative involves correctly identifying a subset of size some c fraction (with c > 0 a constant) to flip. -> negligible probability (in number of BF bits used)

Malicious behavior in OT/Circuit generation

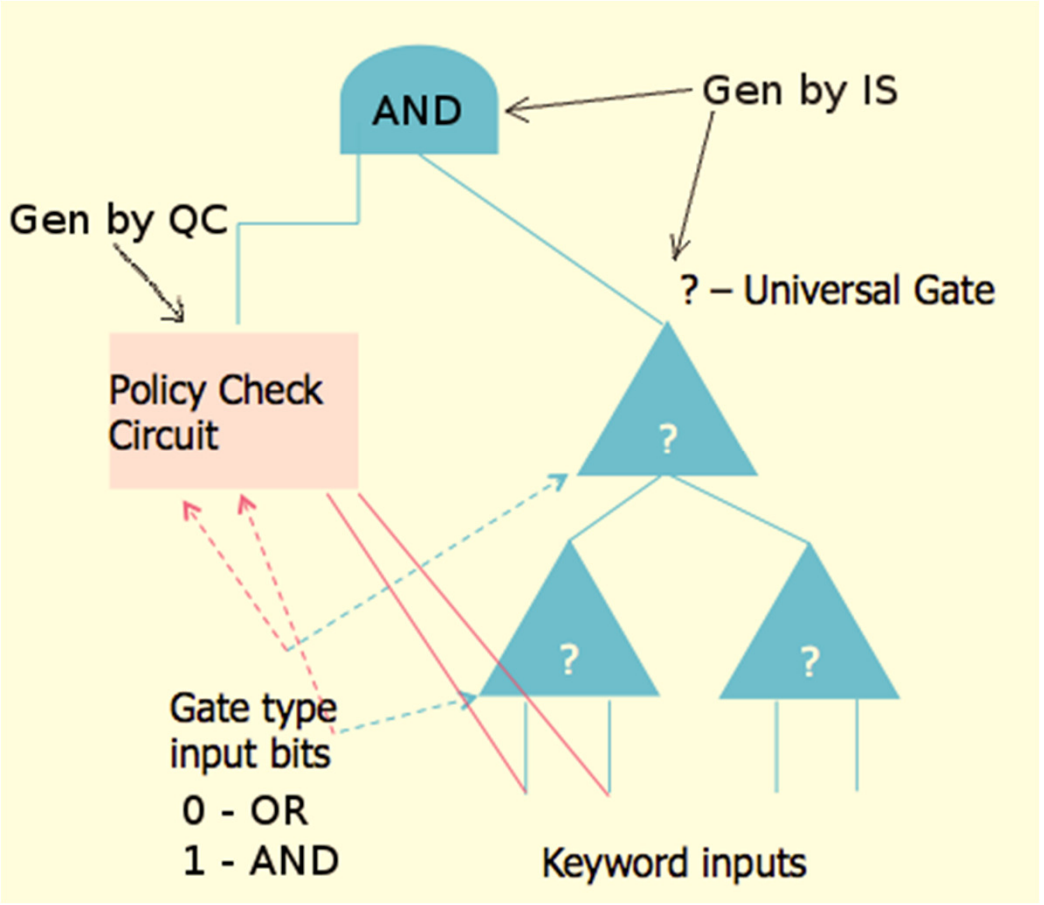


Malicious behavior in OT/Circuit generation

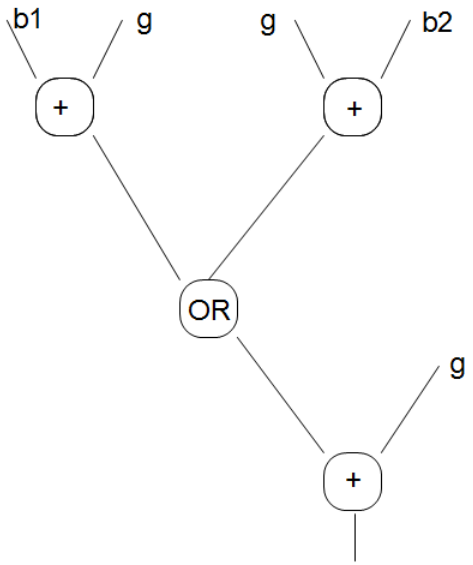


(Malicious..) Universal Circuit

- IS generates universal circuit which evaluates any circuit on any input.
- Circuit has a fixed tree pattern. The gates and inputs are unknown and provided by the client and IS
- AND of policy check circuit and universal circuit performed
- Inputs to PC and Evalualtion Circuits are cryptographically binded.



UC is cheap



If g is 0 (OR gate). then you have b_1 OR b_2

If g is 1 (AND gate), then you have NOT(NOT b_1 OR NOT b_2) = b_1 AND b_2

One non-XOR gate! (XOR gates are free KS08a)

“Practical” circuit for MPC

Unfinished search for MPC benchmark

Beets auction (private circuit?)

AES, DES, Mult, etc. Not clear how useful in “practical MPC”

We give you “The SPAR Circuit”

- Tree
- Pairs of leaves XOR together and then form AND subtrees to evaluate BF
- Output of this is fed into the query formula
  - Usually just a couple of ANDs; a log-domain OR for range or negations.