# *Blind Seer: Bloom Filter Index Search of Encrypted Results*

6th Bar-Ilan Winter School on Cryptography

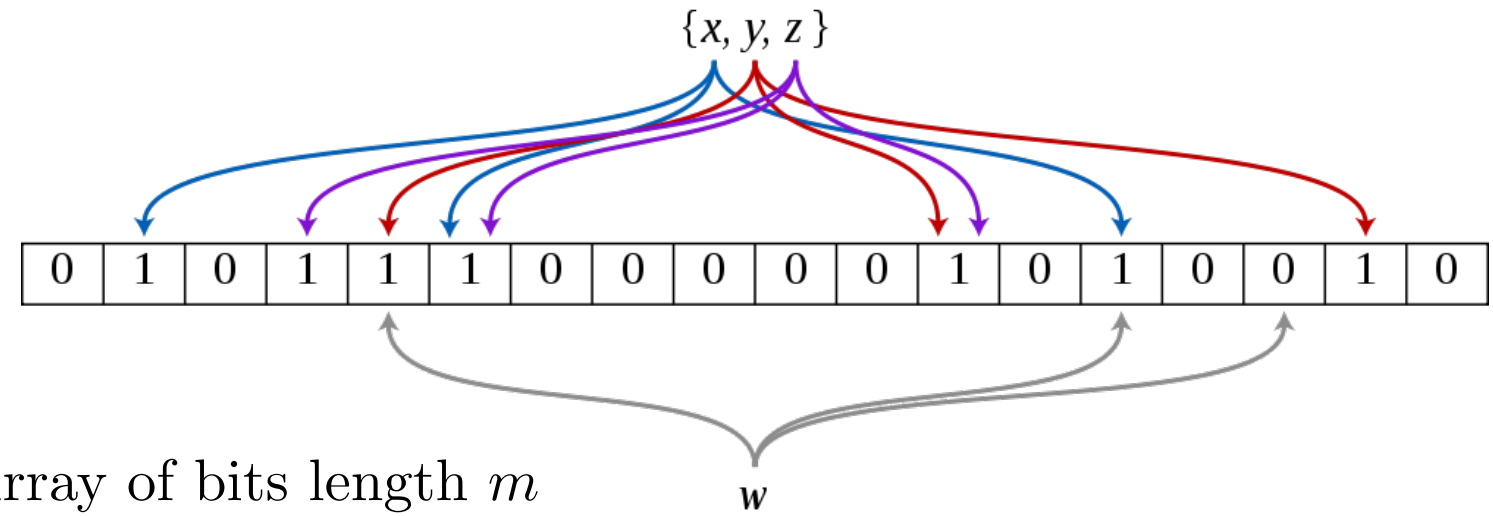
Ben Fisch, Stanford University

# Blind Seer: Bloom Filter Index Search of Encrypted Results


- V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S. G. Choi, W. George, S. M. Bellovin, and A. Keromytis. *Blind Seer: A private scalable DBMS.* In IEEE Symposium on Security and Privacy. IEEE Computer Society, 2014.

- B. A. Fisch, B. Vo, A. Kumarasubramanian, F. Krell, V. Kolesnikov, T. Malkin, S. M. Bellovin. *Malicious-client security in Blind Seer: A scalable private DBMS.* In IEEE Symposium on Security and Privacy. IEEE Computer Society, 2015.

# Bloom Filters




- Array of bits length $m$
- k independent hash functions $H_1, ..., H_k$
- Insert word $x$: set the bits at indices $H_1(x), ..., H_k(x)$
- Lookup word $w$: check the bits at $H_1(w), ..., H_k(w)$
- No false negatives
- $t$ insertions gives false positive probability $\approx (1 - e^{-kt/m})^k$
  - e.g., with $m = 30t$ the false positive rate is $\approx 10^{-6}$

# Masked Bloom Filter

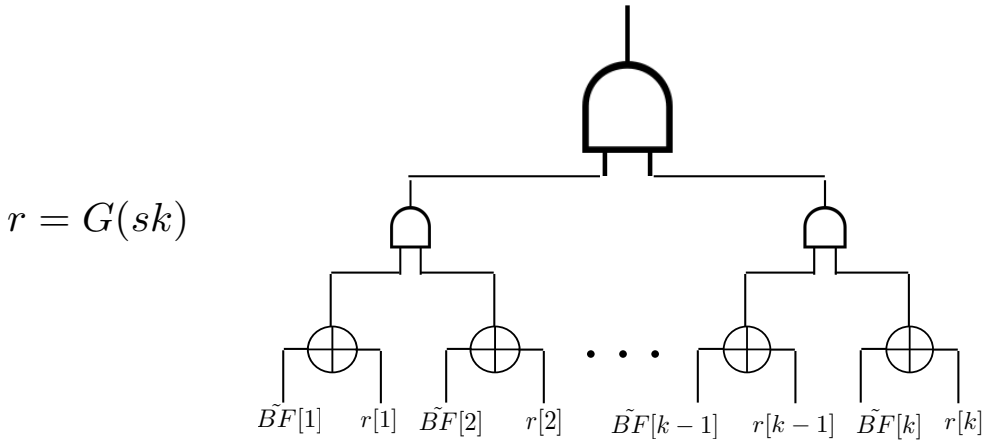
- Client holds a secret key $sk$ and uses a PRG $G$ to generate a pseudorandom pad the length of the Bloom filter, $G(sk) \in \{0,1\}^m$
- Client forms the masked filter $\tilde{BF} = BF \oplus G(sk)$
- Client sends the masked filter $\tilde{BF}$ to the Server

*In the OSPIR setting, the third party Owner forms BF and gives $\tilde{BF}$ to the Server and the key sk to the Client.*

# Private Lookup in Masked BF

- To lookup $w$, Client sends indices $J_w = \{H_1(w), ..., H_k(w)\}$ to Server
- Client and Server run *secure function evaluation* (SFE):
  - Client has $k$ input bits $G(sk)[j]$ for $j \in J_w$
  - Server has $k$ input bits $\tilde{BF}[j]$ for $j \in J_w$
  - The SFE circuit computes $\bigwedge_{j=1}^{k} G(sk)[j] \oplus \tilde{BF}[j]$
  - Only Client learns the output

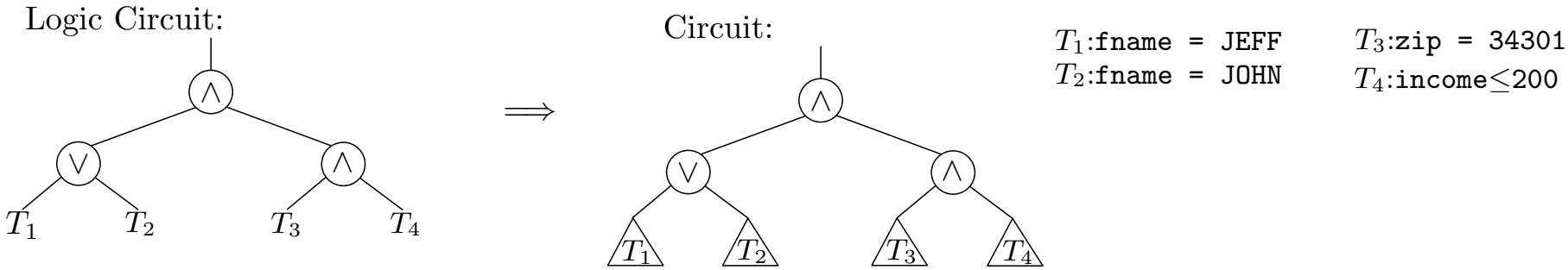
# Garbled Circuit Implementation




- Garbled circuit has $k-1$ non-XOR gates (use free XOR technique)
- GC Protocol:
  - Client garbles: chooses key pairs for all wires including the output wire, replaces gates with ciphertext tables
  - Server evaluates: 1 oblivious transfer (OT) per input, obtains output key from the garbled circuit, sends back to Client
  - Only Client knows the label of the output key
- Secure for semi-honest garbler, malicious evaluator
- Very fast, several $\mu s$ / gate (faster rates for large circuits)

# Private Boolean Queries

`(fname = JEFF OR fname = JOHN) AND zip = 34301 AND income` $\leq$ `200`

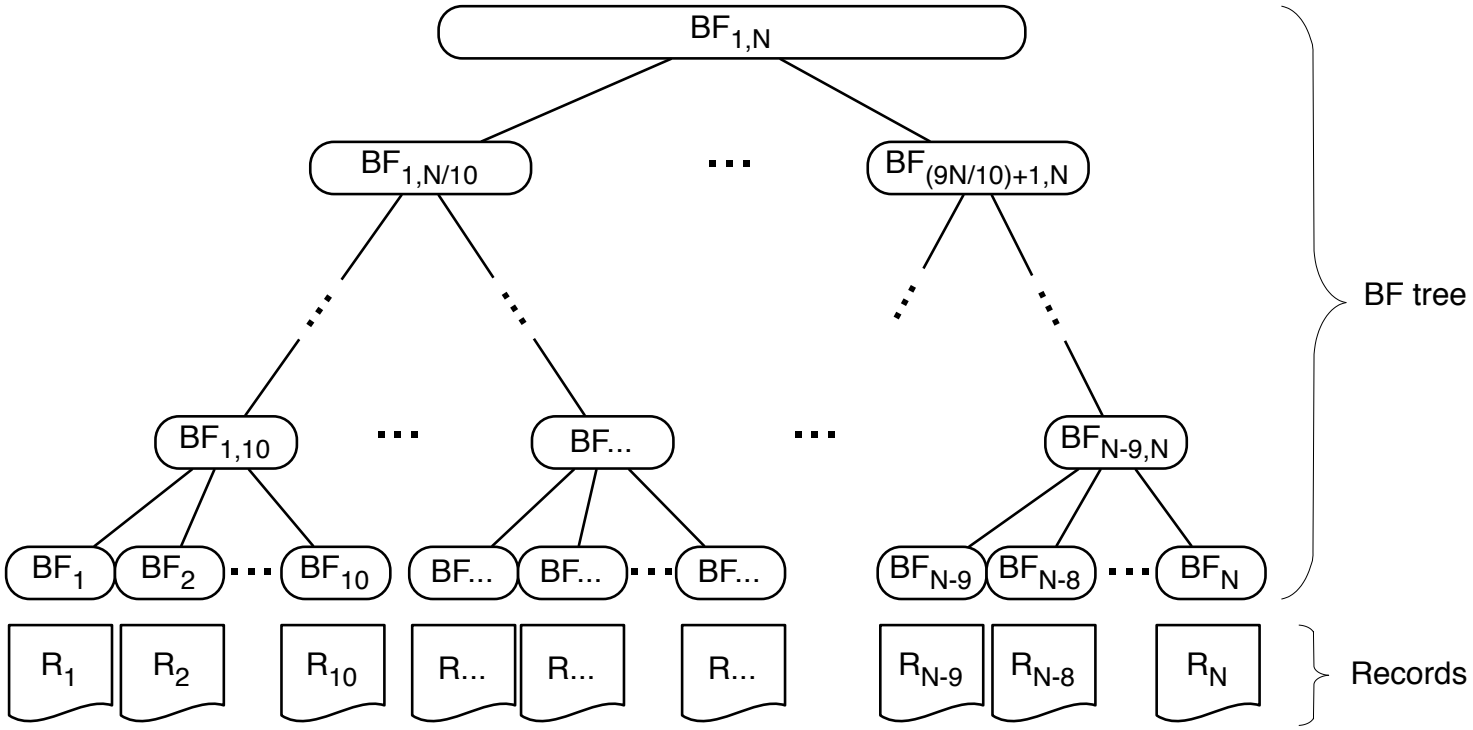



- Boolean expression of Bloom filter single keyword queries
- Each term $T_i$ is replaced with the circuit for a single keyword lookup in a masked BF
- For each keyword $w$, Client sends to Server the indices $H_1(w), ..., H_k(w)$
- Range query terms (e.g. income $\leq 200$) are replaced with a disjunction of $log(N)$ keywords, where $N$ is the maximum size of the range
- The entire circuit is evaluated with SFE

# What is Leaked So Far?

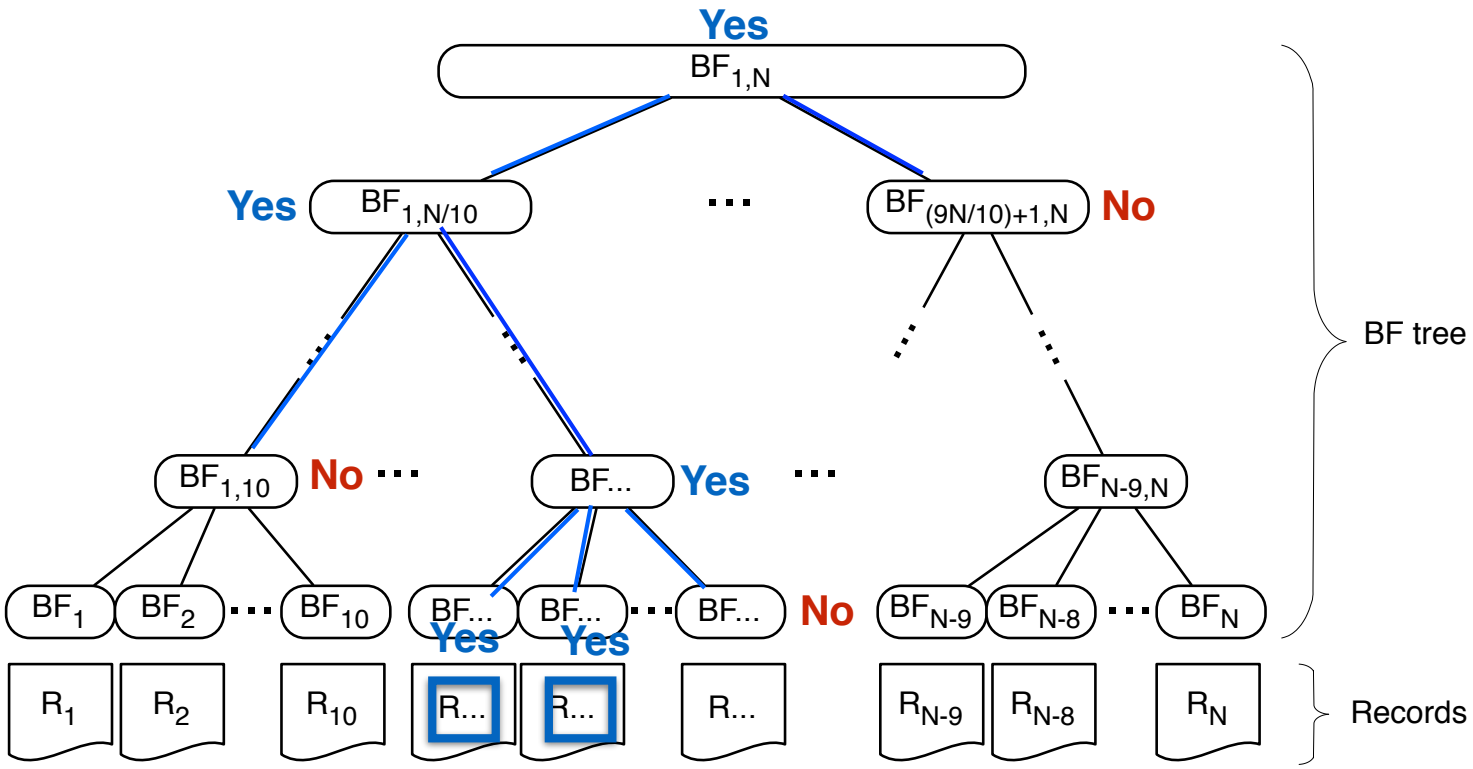
- Keyword hashes
- Repeated queries or keyword terms
- Query logic structure

# Bloom filter Tree Index



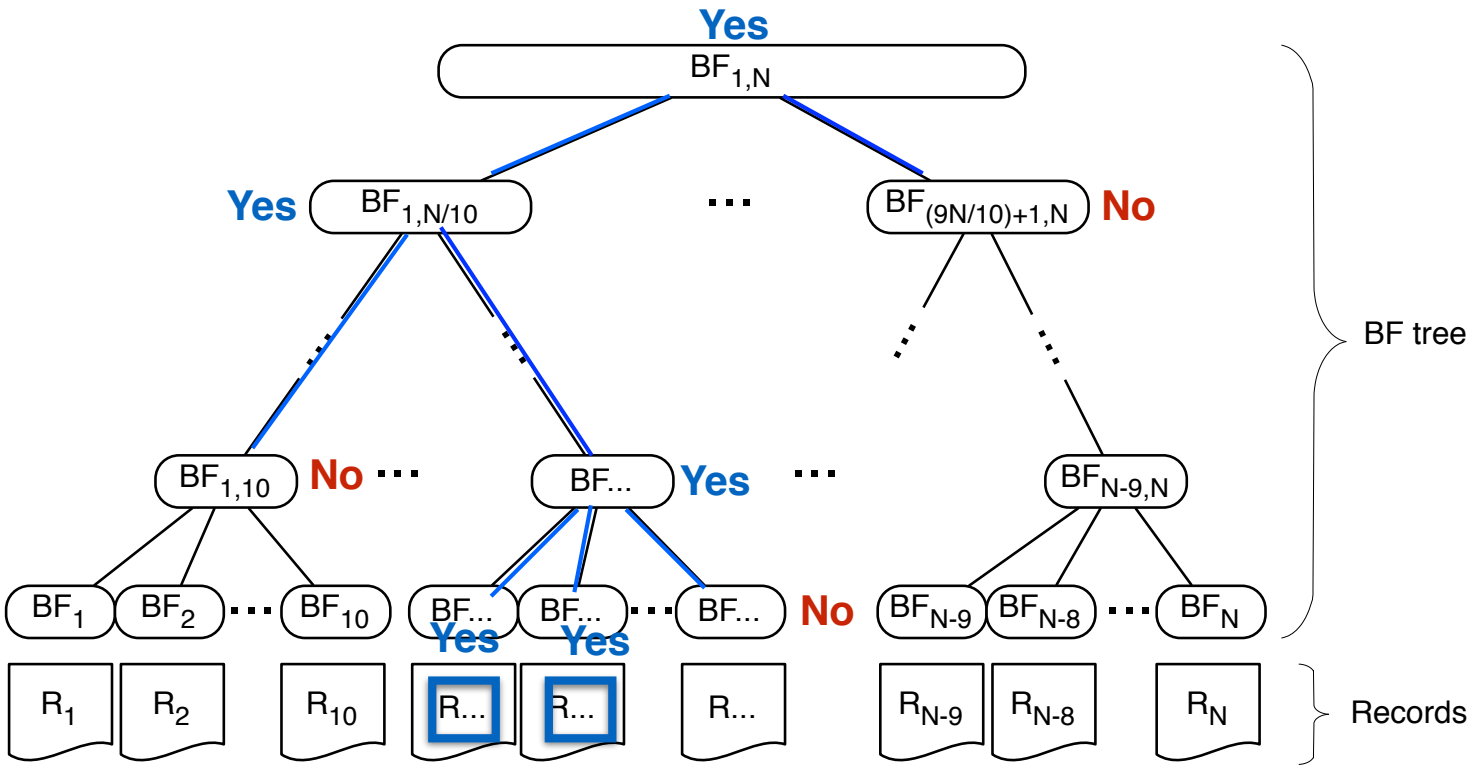

- Each leaf Bloom filter $BF_i$ indexes all the keywords of DB record $R_i$
- Each internal node Bloom filter $BF_{i,j}$ indexes all the keywords of DB records $\{i, ..., j\}$

# DB Search in BF Tree




- Start with evaluating Q on the root BF node.
- If Q returns true on an internal BF node, visit its children. At a leaf node $BF_i$, append $R_i$ to the result set.
- Sublinear search time? For disjunctions, search time is proportional to number of results (asymptotically optimal). For conjunctions, it is at most proportional to number of results for the most selective term.

# DB Search in BF Tree




- This search procedure does not work if Q contains negations. Why?
- Negations (e.g. NOT income $= 200$) are replaced with range terms (e.g. income $< 200$ OR income $> 200$)

# Encrypted BF Tree Index

- The records $R_i$ are permuted randomly and replaced with $\tilde{R}_i = \mathsf{enc}_{s_i}(R_i)$
- Using a pseudorandom function $F$ with secret key $sk$, encrypt each Bloom filter node $BF_v$ as $\tilde{BF}_v = F_{sk}(v) \oplus BF_v$
- In the search procedure, the query $Q$ is evaluated on each node using the protocol for private boolean queries on a masked Bloom filter
- Optimizations: parallelizing the search, batching circuits for SFE, preprocessing OTs

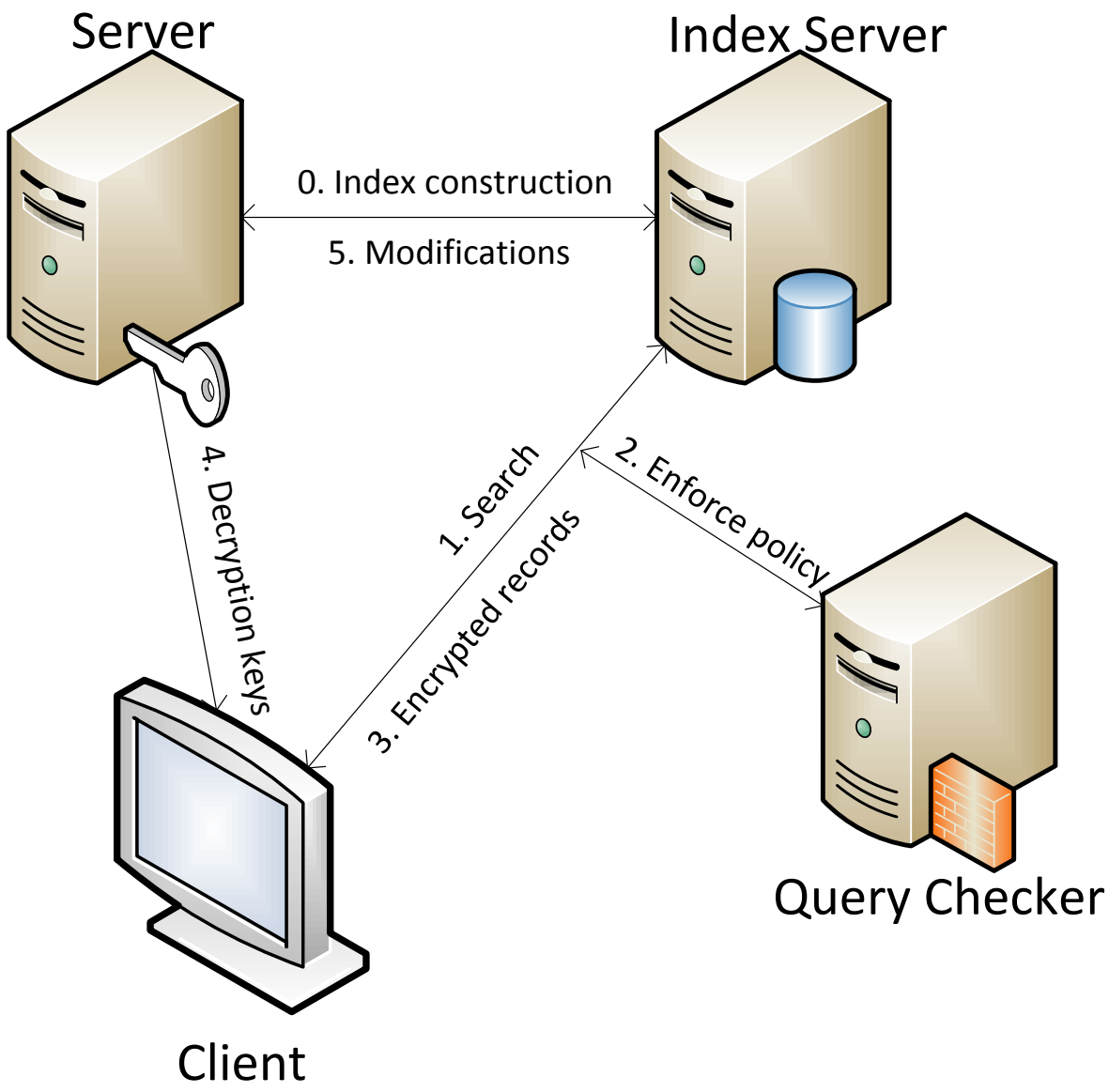
# Leaked Traversal Patterns

- Server sees the search path in the tree index. What information can be inferred?
- Disjunction queries:
  - Equivalent to learning only the result pattern (standard access pattern)
- Conjunction queries:
  - Reveals more information than the result pattern alone, also **abandoned paths**
  - Intersection of traversal patterns of individual terms
  - Less information than learning the result patterns of each individual term?
- May reveal query structure (e.g. disjunction vs conjunction)

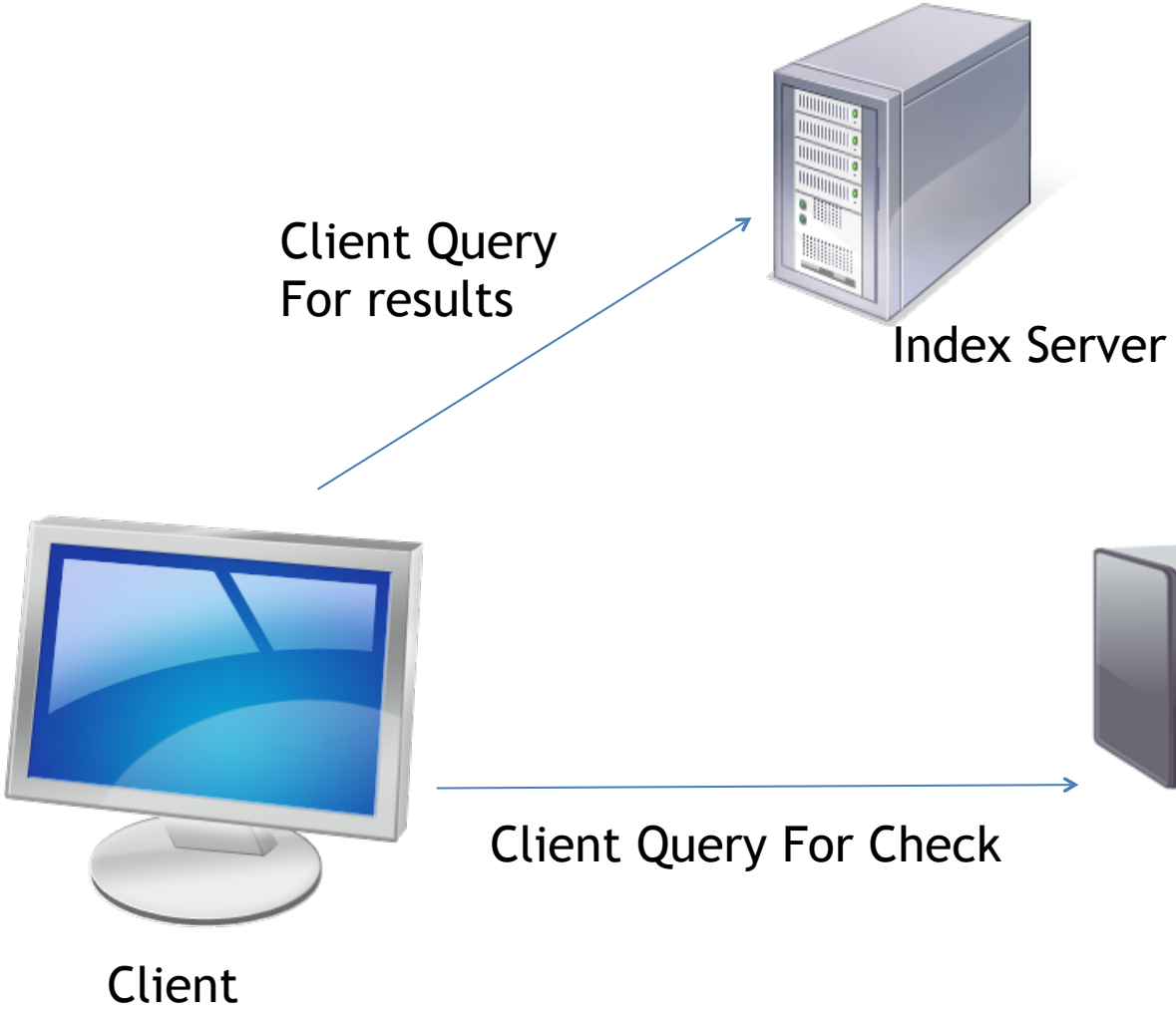
# Security for Semi-honest Server

- The protocol is secure (up to the specified leakage) with a server that behaves semi-honestly
- Example active attack:
  - Server alters mask bit inputs for select keyword terms of the query circuit (will likely cause these terms to return false)
  - May allow Server to learn the traversal pattern for a partial clause or keyword of the query

# Blind Seer Architecture (OSPIR Setting)



# Policy Restricted Queries




**Evaluate policy check and query with separate "Garbled Circuit" protocols**

# Policy Restricted Queries

**semi-honest client**

**Server evaluates garbled Query circuit**

Index Server

**Client generates garbled Query circuit**

Obliviously transfer keys for DB inputs

Client

Obliviously transfer keys for **Query** inputs

Query Checker

**Client evaluates garbled Policy circuit**

**Query Checker generates garbled Policy circuit**

# Malicious Clients

- A malicious client can subvert the policy
- The client can generate an arbitrary query circuit that does not match its inputs to the policy circuit!
- We can fix this cheaply:
  - Make Client the evaluator in both the Policy and Query GC protocols
  - Index Server generates the query circuit using a universal circuit construction (no additional cost due to free XOR)
  - Bind the client's inputs to the two GC protocols by synchronizing the input keys
- Tradeoff: opens more opportunities for an active server attack

*Questions?*

# Universal Circuits

- $UC_k$
  - Simulates any circuit of size k
  - Input description of circuit C of size k and value x, output C(x)

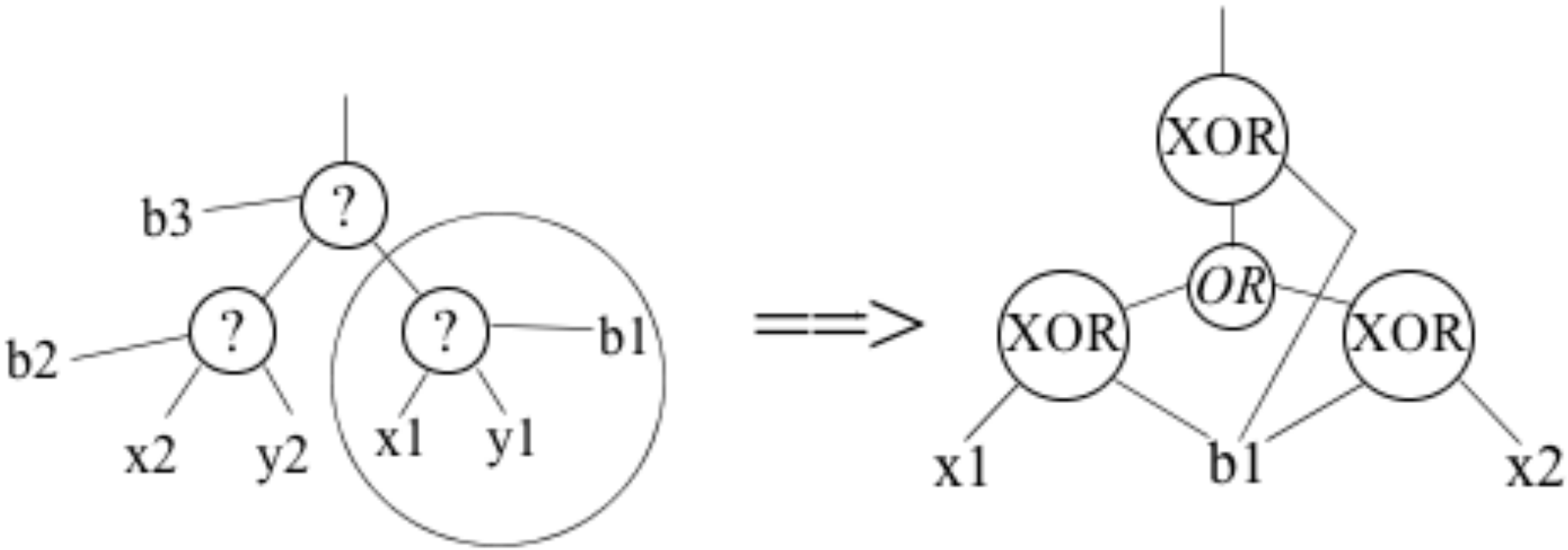
- Garbling $UC_k$ for Private Function Evaluation
  - Hides everything except the circuit size
  - Expensive due to significant circuit size increase

# Universal Query Circuit

- Blind Seer Query circuits
  - Topology is not private (revealed by garbled circuit protocol)
  - Private part of the circuit is monotone (i.e. all gates are AND/OR)

- $UC_T$ – monotone
  - Simulates any monotone circuit with topology
  - Cheap construction
  - **No overhead in GC evaluation using free-XOR**

# Universal Query Circuit Transformation




$$f(x,y,b) = \begin{cases} x \wedge y & b = 1 \\ x \vee y & b = 0 \end{cases} = ((x \oplus b) \vee (y \oplus b)) \oplus b$$

# Malicious-Client Protocol