ESPADA: Efficient Security and Privacy Assurance for Database Access

IARPA SPAR Program Challenge

Stanislaw Jarecki, Gene Tsudik

University of California, Irvine

Craig Gentry, Charanjit Jutla, Hugo Krawczyk,
Michael Steiner, Marcel Rosu

IBM TJ Watson Labs

NIST PEC Workshop

December 8-9, 2011

Talk Outline

1. Private Database Access challenge of the IARPA SPAR program
   - IARPA = Intelligence Advanced Research Projects Activity
   (Research Grant Agency, like DARPA but for Intelligence)
   - SPAR = Security and Privacy Assurance Research (Program)
   ≈ Privacy Assurance in Database Access

2. ESPADA: UC Irvine and IBM proposal for IARPA SPAR Program
   a) Overview
   b) Some of our techniques and efficiency/privacy tradeoffs
   c) Related challenges

Private DB Access: Functionality

Server: Database
Input: **DB**, e.g. Border Crossings

Client: Agent
Input: **Q**, list of queries

Server gets <u>nothing</u>
about Agent's queries **Q**

Client learns **Q(DB)**, i.e.
those records in **DB** which
match queries in **Q**,
and <u>nothing</u> else about **DB**

Private DB Access: Functionality

Server: Database
Input: **DB**, e.g. Border Crossings

Client: Agent
Input: **Q**, list of queries

Attr.1 (first name)	Attr.2 (last name)	Attr.3 (SSN)	Attr.4 (driv.lic. #)	Attr.5 (date)	…	Attr.n (…)
Joe	Smith	712-28-8748	C5121090	2009-07-29	…	…
Joe	Kleinert	418-11-5109	M3109988	2009-07-29	…	…
…	…	…	…		…	…

Private DB Access: Functionality

Server: Database

Input: **DB**, e.g. Border Crossings

Attr.1 (first nm.)	Attr.2 (last nm.)	Attr.3 (SS#)	Attr.4 (licence#)	...	...
Joe	Smith	712-28-8748	C5121090	...	...
Joe	Kleinert	418-11-5109	M3109988	...	...
...	…	...	...	...	...

Client: Agent

Input: **Q**, list of queries

q_1: (last nm = Kleinert)

q_2: (licence# = C5121090)

q_3: ...

Client learns those records in **Q(DB)** and nothing else about Server's **DB**

Server learns nothing about Agent's input **Q**

Private DB Access: Functionality

Server: Database

Input: **DB**, e.g. Border Crossings

Attr.1 (first nm.)	Attr.2 (last nm.)	Attr.3 (SS#)	Attr.4 (licence#)	...	...
Joe	Smith	712-28-8748	C5121090	...	...
Joe	Kleinert	418-11-5109	M3109988	...	...
...	...	...	...	...	...

Client: Agent

Input: **Q**, list of queries

q_1: (last nm = Kleinert)

q_2: (licence# = C5121090)

q_3: ...

Client's Output:					
Joe	Kleinert	418-11-5109	M3109988	...	...

Client learns those records in **Q(DB)** and nothing else about Server's **DB**

Server learns nothing about Agent's input **Q**

Private DB Access: Functionality

Server: Database

Input: **DB**, e.g. Border Crossings

Attr.1 (first nm.)	Attr.2 (last nm.)	Attr.3 (SS#)	Attr.4 (licence#)	...	...
Joe	Smith	712-28-8748	C5121090	...	...
Joe	Kleinert	418-11-5109	M3109988	...	...
...	...	...	...	...	...

Client: Agent

Input: **Q**, list of queries

q_1: (last nm = Kleinert)

q_2: (licence# = C5121090)

q_3: ...

Client's Output:					
Joe	Kleinert	418-11-5109	M3109988	...	...
Joe	Smith	712-28-8748	C5121090	...	...

Client learns those records in **Q(DB)** and nothing else about Server's **DB**

Server learns nothing about Agent's input **Q**

Two phases of the SPAR program

"Baby SPAR": Client retrieves from DB all records matching his queries s.t.:

1. Client learns nothing about the remaining DB records
2. Server learns nothing about Client's queries
3. Simple query type: atomic (single attribute) queries
4. Simple matching function: exact match with specified value
5. Medium DB: 100,000 records, 100KB each
6. Modest efficiency goal: < 100x MySQL

SPAR: the above, extended as follows:

New 3: disjunctions, conjunctions, threshold conj.'s (t-out-of-n matching fields)

New 4: interval ranges, stemming, wildcards, substrings, "close-distance" match

New 5: Large DB: 100,000,000 records, 100KB each

New 6: Stringent efficiency goal: < 10x MySQL

7: dynamic DB (record add, delete, modify)

8: authorization enforcement (on Client's blinded query and credentials)

Private Database Access
General Problem Statement

Server: **DB**
set of records

Client:
query **q** = (at,v)

Server learns
nothing about **q**

Client gets **DB** records
which match query **q**
and nothing else
about **DB**

Fundamental Limitation of PIR:
O(n) work per query to
hide q from Server

Private Database Access
Model Relaxation in ESPADA Scheme

Server: **DB**
set of records

Client:
query **q** = (at,v)

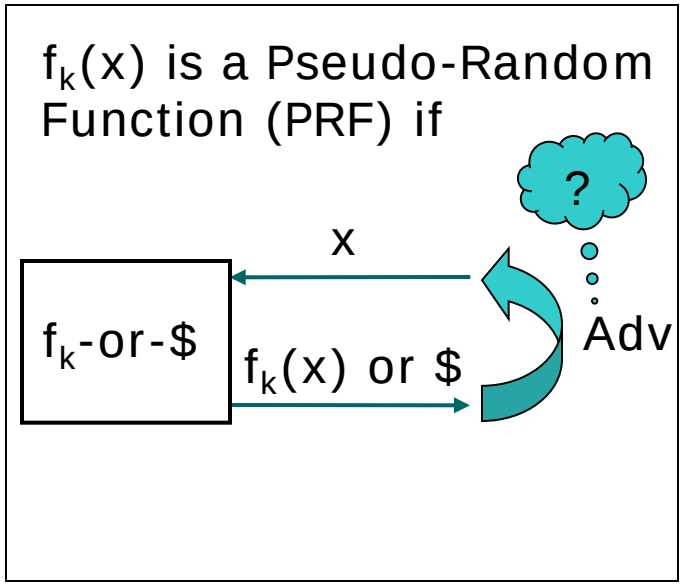
Split Server into two entities:
Server and Proxy

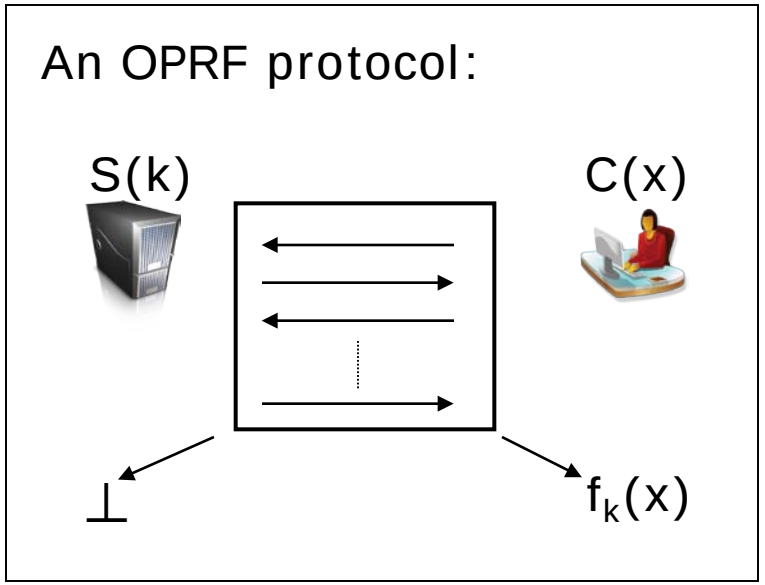
- ✓Server communicates with Proxy only at initialization, sending "Encrypted Database" (**EDB**)
- ✓Goal: O(n) work during initialization
- ✓Goal: O(1) work per query
- ✓Proxy can collude with Client
- ×Server and Proxy collusion breaks Client's privacy
- ×Proxy learns some access pattern information

EDB

Proxy

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





[Yao'82]:	$f_k(x)=AES_k(x)$	O(λ) exp.'s + O(|AES|) sym. ops.	2 msg	DDH *[λ: sec. Par.]*
[NR'04]:	$f_k(x)=g^{\{\Pi k_i \text{ s.t. } x_i=1\}}$	O(λ) exp.'s	2λ msg	DDH
[FIPR'05]:	(same)	O(λ) exp.'s	2 msg	DDH
[JL'09]:	$f_k(x)=g^{\{1/(k+x)\}}$	O(1) exp.'s	2 msg	q-DHI + DCR [Pailier]
[JL'10]:	$f_k(x)=(H[x])^k$	2 exp's	2 msg	"One More DH", ROM

Basic SPAR Solution [JT'09] (Exact Match, Atomic Queries)

I) Initialization: Creating Encrypted DB

Server: **DB**
key k of PRF F_k

Client:
query **q** = (at,v)

EDB: Each record *R* encrypted under $H_1(F_k(at,v))$ for all (at,v) in *R* and *tagged* by $H_2(F_k(at,v),ctr)$ where ctr incremented at each occurence of (at,v)

Proxy: **EDB**

|**EDB**| = |**DB**| + |Tag Table|

Basic SPAR Solution [JT'09] (Exact Match, Atomic Queries)

II) Servicing Client's Queries

Server: **DB**
key k of PRF F_k

Client:
query **q** = (at,v)

"Oblivious PRF": Oblivious Retrieval of $\sigma=F_k(\mathbf{q})$

EDB: Each record *R* encrypted under $H_1(F_k(at,v))$ for all (at,v) in *R* and *tagged* by $H_2(F_k(at,v),ctr)$ where ctr incremented at each occurence of (at,v)

Server learns nothing about **q**

Proxy: **EDB**

|**EDB**| = |**DB**| + |Tag Table|

Basic SPAR Solution [JT'09] (Exact Match, Atomic Queries)

II) Servicing Client's Queries

Server: **DB**
key k of PRF F_k

Client:
query **q** = (at,v)

"Oblivious PRF": Oblivious Retrieval of $\sigma=F_k(\mathbf{q})$

EDB: Each record *R* encrypted under $H_1(F_k(at,v))$ for all (at,v) in *R* and *tagged* by $H_2(F_k(at,v),ctr)$ where ctr incremented at each occurence of (at,v)

Server learns nothing about **q**

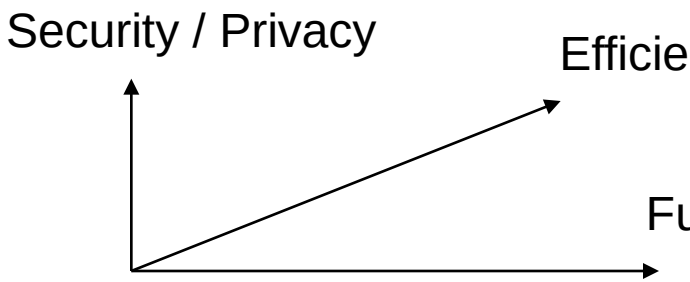
Proxy: **EDB**

|**EDB**| = |**DB**| + |Tag Table|

Look-up (and retrieval) of encrypted records by $H_2(F_k(\mathbf{q}),ctr)$ for ctr=1,2,...

Leaks no information to IB except # of hits (assuming Client caching)

Design Space and Tradeoffs



Provable Security:

Specify exact information leaked to each party

Functionality	Efficiency	Security/Privacy
query types atomic, disj.'s, conj.'s, thr-conj.'s match functions exact, stemming, interval ranges wildcards., substrs., close-dist. record creation static, dynamic (add, delete)	components initialization, retrieval, DB update, authorization resources computation, storage, bandwidth, latency	trust relations Client↔Prx, Server↔Prx fault types HBC, malicious, “covert” info. leakage sizes, access pattern, query type, match fnct., auth. type, DB leakage

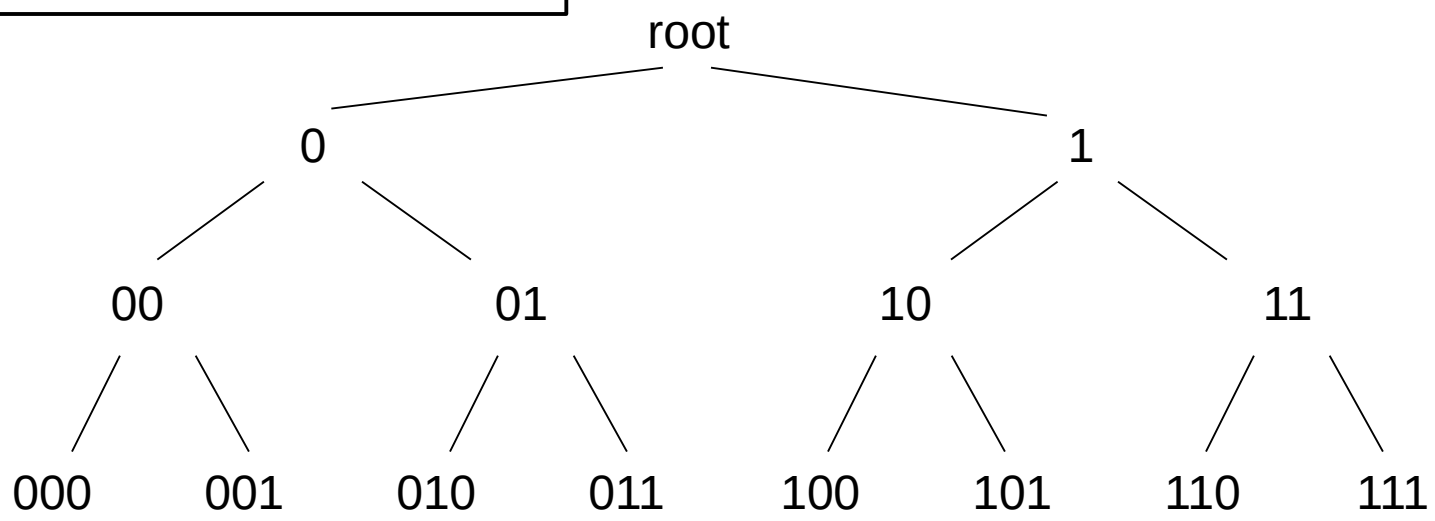
Techniques: (I) Complex Queries
Example: Interval Search

- Matching Function: M(v,q)=1 iff value v matches query q
- E_0,E_1: "expansion functions", mapping v's and q's onto sets of tokens

$$\text{s.t. } E_0(v) \cap E_1(q) = 1 \text{ iff } M(v,q)=1$$

- Initialization: Each R tagged with $F_k(v')$ for all v' in $E_0(v)$
- Retrieval: OPRF retrieves $F_k(q')$ for all q' in $E_1(q)$

values in range $[0,...,2^k)$, here k=3



Techniques: (I) Complex Queries
Example: Interval Search

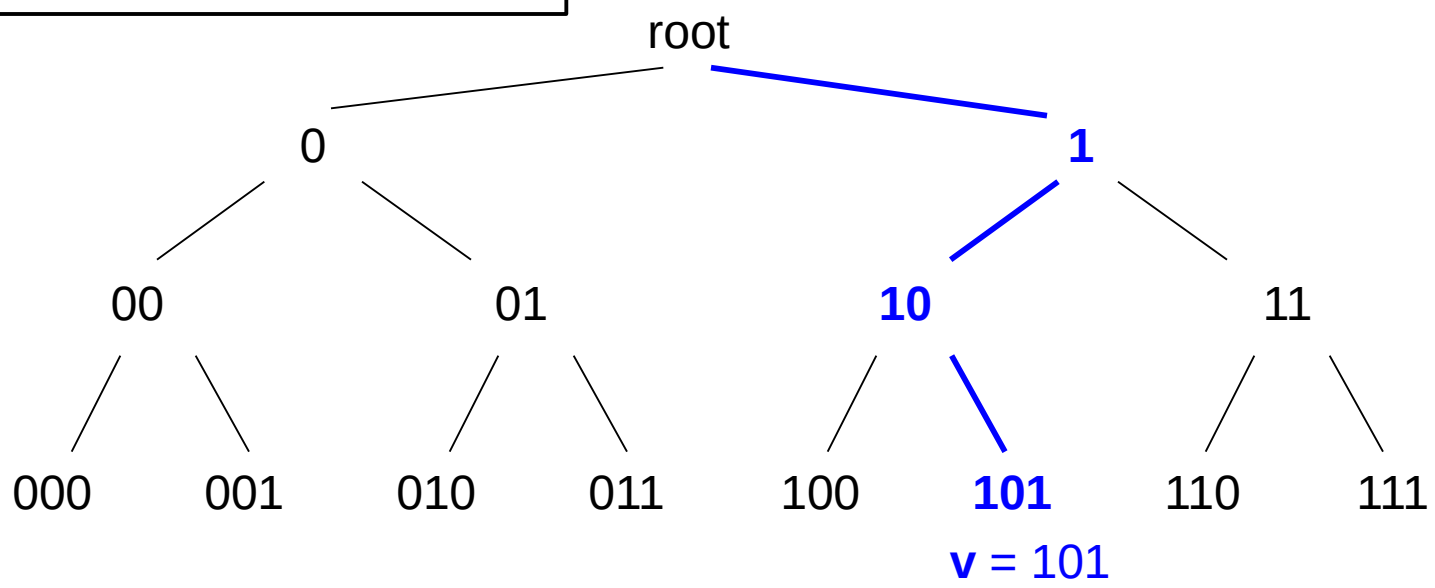
- Matching Function: M(v,q)=1 iff value v matches query q
- E_0,E_1: "expansion functions", mapping v's and q's onto sets of tokens

$$\text{s.t. } E_0(v) \cap E_1(q) = 1 \text{ iff } M(v,q)=1$$

- Initialization: Each R tagged with **$F_k(v')$** for all **v' in $E_0(v)$**
- Retrieval: OPRF retrieves $F_k(q')$ for all q' in $E_1(q)$

values in range $[0,...,2^k)$, here k=3

$E_0(v)$ = { *path from root to v* }



Techniques: (I) Complex Queries
Example: Interval Search

- Matching Function: M(v,q)=1 iff value v matches query q
- E_0,E_1: "expansion functions", mapping v's and q's onto sets of tokens

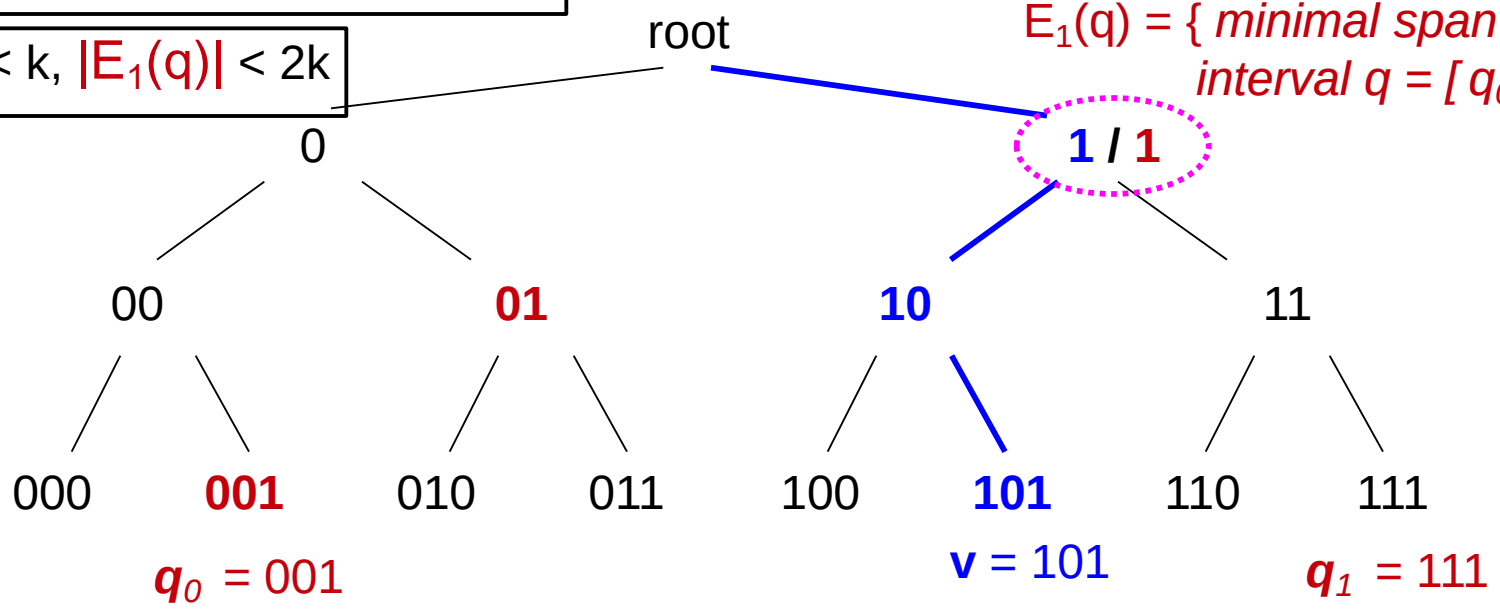
  s.t. $E_0(v) \cap E_1(q) = 1$ iff $M(v,q)=1$
- Initialization: Each R tagged with **$F_k(v')$** for all **$v' \in E_0(v)$**
- Retrieval: OPRF retrieves **$F_k(q')$** for all **$q' \in E_1(q)$**

values in range $[0,...,2^k)$, here k=3

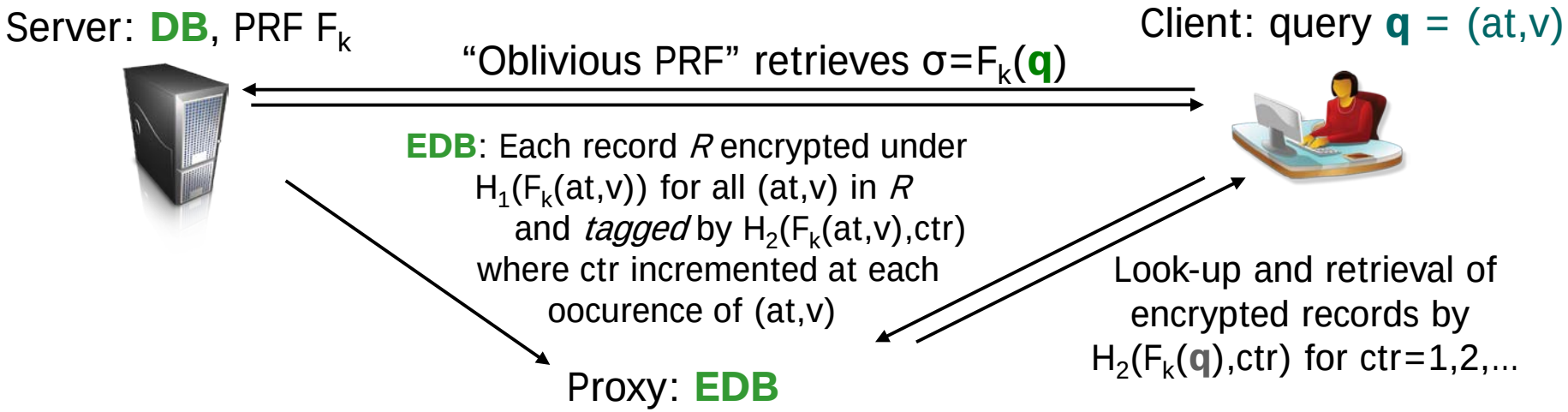
$|E_0(v)| < k$, $|E_1(q)| < 2k$

$E_0(v)$ = { *path from root to v* }

$E_1(q)$ = { *minimal span of interval* $q = [q_0, q_1]$ }



Techniques: (II) Authorizations



- Adding ZK Proof to OPRF:

  Client proves in ZK that it's query satisfies Server's access policy

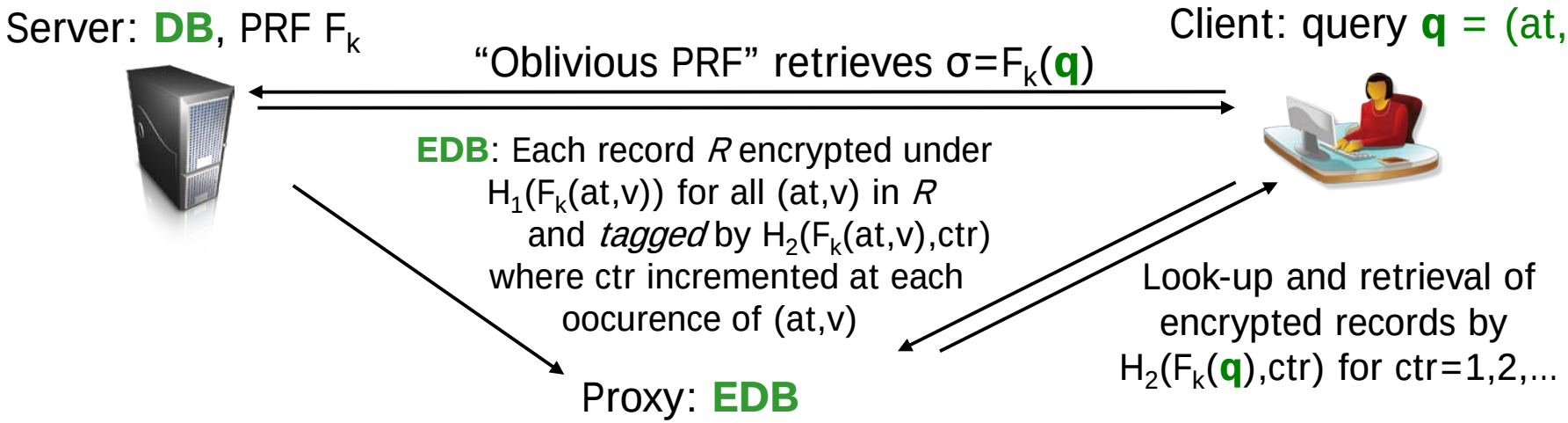
- Efficiency depends on access policy complexity:

  restrictions on attributes; query types; match functions; query values?

- Efficiency depends on Client's privacy protection:

  can Server learn query attribute, query type, match function ?

Techniques: (III) Conjunctions



Idea 1: Identify (encrypted) record sets matching each term and use MPC for Set Intersection on encrypted sets

O(|largest set|) public key ops; Proxy learns these (encrypted) sets

Idea 2: a) EDB stores mac's $F_k(R,(at,v))$ binding R to (at,v) pairs;

b) Proxy retrieves mac's binding same R to each query term;

c) Client obliviously exchanges these mac's for decryption of R.

O(|intersection|) p.k. ops; Proxy learns sets matching conjunctions of terms

Idea 3,4,...: Refinements for other efficiency/privacy trade-offs

Private Database Access
Some Research Challenges

- Many cryptographic protocol problems related to updates, authorizations, conjunction-handling, …
  - Partial List: OPRF, set intersection on encrypted items, escrow, malicious/covert security (overhead of ZK proofs)
- Privacy-Preserving Information Retrieval algorithms:
  IR algorithms that perform only equality tests? (cf. range queries)
- Can order-preserving encryption provide any gains?
- Use dedicated secure two-party computation, e.g. AES-OPRF?
- Use Oblivious RAM to hide patterns in queries and updates?
- Quantifying privacy leakage: differential privacy techniques?