



INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

Outsourced Data By Using Hamming Joins

Vijay J. Chaudhari

* B.E.Computer,Professor, Sandip Institute of Polytechnic,Nashik, India.

Abstract

In this paper we introduce a mechanism for executing general binary JOIN operations (for predicates that satisfy certain properties) in an outsourced relational database framework with computational privacy and low overhead – a first, to the best of our knowledge. In an outsourced database framework, clients place data management responsibilities with specialized service providers. Of essential concern in such frameworks is data privacy. Potential clients are reluctant to outsource sensitive data to a foreign party without strong privacy assurances beyond policy “fine prints”.

We experimentally evaluate the main overhead components and show they are reasonable.

We illustrate via a set of relevant instances of JOIN predicates, including: range and equality (e.g., for geographical data), Hamming distance (e.g., for DNA matching) and semantics (i.e., in health-care scenarios – mapping antibiotics to bacteria). The initial client computation overhead for 100000 data items is around 5 minutes and our privacy mechanisms can sustain theoretical throughputs of several million predicate evaluations per second, even for an un-optimized Open SSL based implementation.

Keywords: Data Encryption, Security and Privacy Protection.

Introduction

Outsourcing the “database as a service emerged as an affordable data management model for parties (“data owners”) with limited abilities to host and support large in-house data centers of potentially significant resource footprint. In this model a *client* outsources its data management to a *database service provider* which provides online access mechanisms for querying and managing the hosted data sets.

Because most of the data management and query execution load is incurred by the service provider and not by the client, this is intuitively advantageous and significantly more affordable for parties with less experience, resources or trained manpower. Compared with e.g., a small company, with likely a minimal data management knowledge, such a database service provider intuitively has the advantage of expertise and the ability to offer the service much cheaper, with increased service availability and uptime guarantees.

Significant security issues are associated with such “out-sourced database” frameworks, including communication-layer security and data confidentiality. Confidentiality can be achieved by encrypting the outsourced content. Once

encrypted however, the data cannot be easily processed by the server. This limits the applicability of outsourcing, as the type of processing primitives available will be reduced dramatically.

Thus, it is important to provide mechanisms for server-side data processing that allow both confidentiality and a sufficient level of query expressibility. This is particularly relevant in relational settings. Recently, protocols for equijoin and range queries have been proposed.

Here we go one step further and provide low overhead solutions for *general* binary JOIN predicates that satisfy certain properties: for any value in the considered data domain, the *number* of corresponding “matching” pair values (for which the predicate holds) is upper bound. We call these finite match predicates (FMPs).

Materials and methods

We choose to keep the data outsourcing model concise yet representative. Sensitive data is placed by a client on a database server situated at the site and under the control of a *database service provider*. Later, the client can access the outsourced data

through an online query interface exposed by the server. Network layer confidentiality is assured by mechanisms such as SSL/IPSec. For this purpose, they will encrypt data before outsourcing. As encrypted data is hard to process without revealing it, to allow for more expressive server-side data processing, clients will also pre-process data according to a set of supported (join) predicates. They will then outsource additional associated metadata to aid the server in processing tasks. This metadata, however, will still be “locked” until such processing tasks are requested by the client. Later, to allow server-side data processing, the client will provide certain “unlocking” information for the metadata associated with the accessed items. The server will perform *exactly* the considered query (and nothing more) without finding out any additional information.

It is important for the outsourced metadata not to reveal any information about the original data. Additionally, the computation, storage and network transfer overhead should maintain the cost advantages of outsourcing, e.g., execution times should not increase significantly.

Finite Match Predicates (FMPs)

In this paper we consider binary predicates $p : X \times Y \rightarrow B = \{\text{true}, \text{false}\}$ for which the “match sets” $P(x) := \{y | p(x, y) = \text{true}\}$ can be computed by a polynomial time algorithm and their size (taken over all encountered values of x) is upper bound. In other words, given a certain value x in the considered data domain, its “matching” values can be determined in polynomial time and their number is upper bound. We call these predicates *finite match* predicates (FMPs). For a relation R matched against a relation S , we define MMS, the *maximum match size*, to be the maximum number of matching values from relation R for any row in relation S . For instance, consider the following discrete time – range join query that joins arrivals with departures within the same 30 mins interval (e.g., in a train station):
SELECT * FROM arrivals,departures WHERE ABS(arrivals.time - departures.time) < 30
In this example, the FMP has an MMS of 60.

Privacy Requirements

In the considered adversarial model, the following privacy requirements are of concern.

Initial Confidentiality

The server should not be able to evaluate inter-column join predicates on *initially* stored data without client “unlock” permission. Formally, given a relation A with encoded elements $D[a_1], \dots, D[a_n]$,

a relation B with encoded elements $D[b_1], \dots, D[b_m]$, any random values $i \in \{1 \dots n\}$ and $j \in \{1 \dots m\}$, for any probabilistic polynomial time server algorithm S , the value $|P r[S(D[a_i], D[b_j])] - 1/2|$ is negligible.

Predicate Safety. Following a client join request, the server can only evaluate the stored data for the predicate provided by the client. Specifically, given a relation A with encoded elements $D[a_1], \dots, D[a_n]$, a relation B with encoded elements $D[b_1], \dots, D[b_m]$, and a predicate pred for which the client provides opening information $\text{open}(\text{pred})$, the server can only learn the value $\text{pred}(a_i, b_j) \in \{\text{true}, \text{false}\}, \forall i = 1 \dots n$ and $j = 1 \dots m$. Formally, given a predicate pred and corresponding

$\text{open}(\text{pred})$ revealed by the client, for any other predicate $\text{pred}' \neq \text{pred}$ for which the server does not have $\text{open}(\text{pred}')$ and any random values $i \in \{1 \dots n\}$ and $j \in \{1 \dots m\}$, for any probabilistic polynomial time server algorithm S , the value $|P r[S_{\text{pred}'}(\text{open}(\text{pred}), D[a_i], D[b_j])] - 1/2|$ is negligible.

We stress that here we do not provide confidentiality of predicates, but rather just of the underlying target data. We also note that we do not consider here the ability of the server to use out of band information and general knowledge about the data sets to infer what the underlying data and the query results look like. In fact we envision a more formal definition in which privacy guarantees do not allow any leaks to the server beyond exactly such inferences that the curious server may do on its own based on outside information.

Performance Constraints

The main performance constraint we are interested in is *maintaining the applicability of out-sourcing*. In particular, if a considered query load is more efficient (than client processing) in the unsecured data out-sourcing model – then it should still be more efficient in the secured version. We believe this constraint is essential, as it is important to identify solutions that validate in real life. There exist a large number of apparently more elegant cryptographic primitives that could be deployed that would fail this constraint. In particular, experimental results indicate that *predicate evaluations on the server should not involve any expensive (large modulus) modular arithmetic such as exponentiation or multiplication*. We resisted the (largely impractical) trend (found in existing research) to use homomorphisms in server side operations, which would have simplified the mechanisms in theory but would have failed in

practice due to extremely poor performance, beyond usability. In fact, in Section V we show that solutions that would employ homomorphisms would be several (2-4) orders of magnitude slower than solutions that we propose in this paper.

We assume that server storage is cheap. This assumption is supported by recent findings that show the total cost of storage management is orders of magnitude higher than the storage equipment acquisition costs.

Adversary

We consider an *honest but curious* server: given the possibility to get away undetected, it will attempt to compromise data confidentiality (e.g., in the process of query execution). The protocols in this paper are protecting mainly data confidentiality. The server can certainly choose to deny service by explicitly not cooperating with its clients, e.g., by not returning results or simply closing connections.

Tools

Encryption, Hashing and Random Numbers.: We consider ideal, collision-free hashes, denoted by H . We consider semantically secure (IND-CPA) encryption mechanisms. We denote by $E_K(v)$ the encryption of value v with secret key K . If not specified, the key K will be implicitly secret and known only to the client. In the following, we use the notation $x \xrightarrow{R} S$ to denote x 's uniformly random choice from S .

p	prime number
N	bit size of p
G	subgroup of Z_p
ρ	order of G
g	generator of G
x_A, y_A	secret values for column A

TABLE OF SYMBOLS USED IN OUR SOLUTIONS.

Proof: Let us assume that for a relation A with encoded elements $D[a_1], \dots, D[a_n]$ and a relation B with encoded elements $D[b_1], \dots, D[b_m]$, there exists a PPT algorithm A and a pair of values $i \in \{1..n\}$ and $j \in \{1..m\}$ such that $|Pr[S(D[a_i], D[b_j]) = 1/2] - 1/2| > \epsilon$. Let the element $D[a_i] = [E_K(a_i), O(a_i), BF(a_i)]$ and let $D[b_j] = [E_K(b_j), O(b_j), BF(b_j)]$. Then, A can have advantage ϵ only if (i) $E_K(a_i)$ can be distinguished from $E_K(b_j)$ with advantage larger than ϵ , or if (ii) $O(a_i)$ can be distinguished

from $O(b_j)$ with advantage larger than ϵ or if (iii) $O(a_i)$ can be searched for in $BF(b_j)$ (the symmetric case is identical). In case (i), we can also build an algorithm that has advantage larger than ϵ against the IND-CPA game of the semantically secure encryption E . Case (ii) cannot occur in an information theoretic sense, since the values $O(a_i)$ and $O(b_j)$ are obfuscated with different random values. For case (iii), let us consider for simplicity that $BF(b_j)$ stores the set $P(b_j)$ as the set of obfuscated values $e_B(v)$, where $v \in P(b_j)$ – instead of using a Bloom filter to encode the $e_B(v)$ values. Then, if A can find $g^{O(a_i)}$ in the set of values $e_B(v) = g^{H(v)y^B}$, then we can also build an algorithm that defeats the discrete logarithm assumption.

Algorithm : The J_H algorithm performing a Hamming join between columns A and B .

```

-----
hamming JOIN(A, B, r_A(k), r_B(k), r_k, k =
1..β) forall a_i ∈ A and k = 1..β do
    v(a_ik) = r_A(k)^{Z(a_ik)}
    mod p; forall b_j ∈ B and
    k = 1..β do
        v(b_jk) = r_B(k)^{Z(b_jk)}
        mod p; u(b_jk) =
        r_k^{O_k(b_jk)} mod p;
    forall b_j ∈ B
    do forall
        a_i ∈ A do
            c ← 0;
            for (k ← 1; k ≤ β; k ←
            k + 1) if v(a_ik) ≠
            v(b_jk) then
                if BF_{ij}(A).contains(u(b_jk)) then
                    c ← c + 1;
                else
                    c ← -1;
                    #signal drop break;
            if c = -1 then continue; #drop(a_i, b_j) if
            c ≤ d then output[E_K(a_i), E_K(b_j)];
-----
    
```

Results and discussion

We now prove the following result for the Hamming join solution proposed above.

Predigate Safety Notes:

Prdigate Safety: The server can in fact determine the actual Hamming distance between matching (but encrypted) (a_i, b_j) pairs (satisfying the $d_H(a_i, b_j) < d$ condition). Moreover, the server can also find the Hamming distance of some encrypted (a_i, b_j) pairs for which $d < d_H(a_i, b_j) \leq \beta$. While out of scope here,

a solution can be provided for this case by prefixing original a_i, b_j values with a random number of special symbols with controllable Hamming distances.

Complexity Analysis: Let T_{encr} be the time to encrypt an element, T_{exp} the time to perform one modular (p) exponentiation, T_{mul} the time to perform a modular (q) multiplication and T_{hash} the time to perform a crypto-hash operation. h is the number of hash functions used to encode elements in a Bloom filter. Then, if t is the number of attributes in the relation, the following results hold.

Lemma: The initial client overhead is $tn(T_{\text{encr}} + 2\beta(T_{\text{exp}} + T_{\text{hash}} + T_{\text{mul}}) + b(T_{\text{exp}} + (h + 1)T_{\text{hash}} + T_{\text{mul}}))$.

Proof: The per-element initial overhead is the sum of three factors:

- (i) the cost to encrypt the element,
- (ii) the cost to generate the obfuscated O and Z values and
- (iii) the cost to generate the β Bloom filters, each storing b/β elements

The cost of storing one element in a Bloom filter is equal to the cost of generating the obfuscated element (a crypto-hash application and an XOR) plus the cost of another h crypto-hashes for generating the bit-wise positions to be set to 1.

Arbitrary Alphabets

The above solution can also be deployed for an arbitrary alphabet, that is, when the elements stored in the database D are composed of symbols from multi-bit alphabets (e.g., DNA sequences). This can be done by de-ploying a custom binary coding step. Let $A = \{\alpha_0, \dots, \alpha_{u-1}\}$ be an alphabet of u symbols. In the pre-processing phase, the client represents each symbol over u bits ($u/\log u$ -fold blowup in storage), such that symbol $\alpha_u = 2^i$. That is, $d_H(\alpha_i, \alpha_j)$ is 1 if $i \neq j$ and 0 otherwise. If each data item has b symbols, each of the item's blocks will have bu/β bits, and, due to the coding, pairs of elements of symbol-wise distance d will have a $2d$ bit-wise Hamming distance. Thus, after the coding phase, the above algorithm can be deployed without change. As an example, for an alphabet of 4 symbols $\{A, C, G, T\}$, the following encoding will be used $\{A=0001, C=0010, G=0100, T=1000\}$. To compare the strings ACG and ACT (alphabet distance 2), the following two binary strings will be compared instead: 000100100100 and 000100101000 (binary Hamming distance 2).

Arbitrary Distances.: One drawback of the previous solution is the fixed nature of the Hamming distance d that can be considered. To accommodate a different distance, additional metadata would need to

be generated by the client accordingly. Instead, it would be desirable to provide a single solution for any distances. In the following we show how to extend the above solution for arbitrary distances.

For this purpose, the encoding algorithm E_H is modified to perform a hierarchical generalization of the previous shuffle-and-divide pre-processing step.

Conclusion

In this paper we introduced mechanisms for executing JOIN operations on outsourced relational data with full computational privacy and low overheads. The solution is not hard-coded for specific JOIN predicates (e.g., equijoin) but rather works for a large set of predicates satisfying certain properties. We evaluated its main overhead components experimentally and showed that we can perform more over 5 million private FMPs per second, which is between two and four orders of magnitude faster than alternatives that would use asymmetric encryption algorithms with homomorphic properties to achieve privacy.

Acknowledgements

We would like to thank the anonymous reviewers for their detailed feedback.

References example:


- [1] E. Bertino, S. Jajodia, and P. Samarati. A flexible authorization mechanism for relational data management systems. *ACM Transactions on Information Systems*, 17(2), 1999.
- [2] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.
- [3] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Proceedings of Eurocrypt 2004*, pages 506–522. LNCS 3027, 2004.
- [4] A. Broder, M. Mitzenmacher, and A. B. I. M. Mitzenmacher. Network applications of bloom filters: A survey. In *Internet Mathematics*, pages 636–646, 2002.
- [5] *Biometrix Int.* <http://www.biometrix.at/>.
- [6] *International HapMap Project.* <http://www.hapmap.org/>.
- [7] *TWIRL and RSA Key Size.* Online at <http://www.rsasecurity.com/rsalabs/node.asp?id=2004>.
- [8] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Hippocratic databases. In *Proceedings of the International Conference on*

Very Large Databases VLDB, pages 143–154, 2002.

R. Agrawal and R. Srikant. Privacy-preserving data mining. In Proceedings of the ACM SIGMOD, pages 439–450, 2000.

[9] *C. Clifton, M. Kantarcioglu, A. Doan, G. Schadow, J. Vaidya, A. Elma-garmid, and D. Suciu. Privacy-preserving data integration and sharing. In The 9th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery, pages 19–26. ACM Press, 2004.*

Author Bibliography

	<p>Vijay Chaudhari Lecturer at Sandip Institute of Polytechnic, Nashik Email: vijay.chaudhari@sipnashik.org</p>
---	--