Scaling Cryptographic Techniques by Exploiting Data Sensitivity at a Public Cloud*

Sharad Mehrotra,[1] Shantanu Sharma,[1] Jeffrey D. Ullman[2]
[1] University of California, Irvine, USA. [2] Stanford University, USA.

ABSTRACT

Despite extensive research on cryptography, secure and efficient query processing over outsourced data remains an open challenge. This poster continues along the emerging trend in secure data processing that recognizes that the entire dataset may not be sensitive, and hence, non-sensitivity of data can be exploited to overcome some of the limitations of existing encryption-based approaches. In particular, this poster outlines a new secure keyword search approach, called *query keyword binning* (QB) that allows non-sensitive parts of the data to be outsourced in clear-text while guaranteeing that no information is leaked by joint processing of non-sensitive data (in clear-text) and sensitive data (in encrypted form). QB improves the performance of and strengthens the security of the underlying cryptographic technique by preventing size, frequency-count, and workload-skew attacks.

CCS CONCEPTS

• **Security and privacy → Database and storage security**.

KEYWORDS

Cryptographic techniques; scalability.

ACM Reference Format:
Sharad Mehrotra, Shantanu Sharma, Jeffrey D. Ullman. 2019. Scaling Cryptographic Techniques by Exploiting Data Sensitivity at a Public Cloud. In *Ninth ACM Conference on Data and Application Security and Privacy (CODASPY '19), March 25–27, 2019, Richardson, TX, USA.* ACM, NY, NY. 3 pages. DOI: https://doi.org/10.1145/3292006.3302384

1 INTRODUCTION

The numerous benefits of public clouds (*e.g.*, no cost of purchasing, installing, running, maintaining data management systems) impose significant security and privacy concerns related to sensitive data storage (*e.g.*, sensitive client information, credit card, social security numbers, and medical records) or the query execution. Such concerns are not a new revelation – indeed, they were identified as a key impediment for organizations adopting the database-as-as-service model in early work on data outsourcing [13]. Since then, security/confidentiality challenge has been extensively studied in both the cryptography and database literature, which has resulted in many techniques to achieve *data privacy*, *query privacy*, and *inference prevention*. Existing work on secure data/query outsourcing may loosely be classified into the following three categories:

(1) **Encryption based techniques.** *E.g.*, order-preserving encryption [1], deterministic encryption, homomorphic encryption [10], bucketization [13], searchable encryption [22].
(2) **Secret-sharing [20] based techniques.** *E.g.*, distributed point function [11], function secret-sharing [4], accumulating-automata [7, 8], and others [9].
(3) **Trusted hardware-based techniques.** They are either based on a secure coprocessor (*e.g.*, [2]) or Intel SGX [5], which allows decrypting data in a secure area and perform some computations (*e.g.*, Opaque [24]).

Techniques	Time	Preventing attacks		
		Size	Workload-Skew	Access-patterns
Deterministic encryption	$1.43x$			
Non-deterministic encryption [18]	$2.1x$			
DSSE [14]	$3281x$			✓
SGX [5]	$6724x$			✓
Full-retrieval	$11135x$	✓	✓	✓
Homomorphic Encryption with ORAM	$> 11135x$			✓

Table 1: Comparing different cryptographic techniques in terms of time (for selection queries over TPC-H data) and attacks. x is the time to search a predicate in cleartext. ✓indicates a technique is not vulnerable to a given attack.

Existing solutions suffer from several limitations. First, cryptographic approaches that prevent leakage, *e.g.*, fully homomorphic encryption coupled with ORAM, simply do not scale to large data sets and complex queries for them to be of practical value. Most of the above-mentioned techniques are not developed to deal with a large amount of data and the corresponding overheads of such techniques can be very high (see Table 1 comparing the time taken for TPC-H selection queries under different cryptographic solutions; note that none of the above-mentioned techniques are completely secure against each attack mentioned in the table, except the full retrieval of the database from the public cloud to the trusted private side). Second, systems such as CryptDB [19] have tried to take a more practical approach by allowing users to explore the tradeoffs between the system functionality and the security it offers. Unfortunately, precisely characterizing the security offered by such systems given the underlying cryptographic approaches have turned out to be extremely difficult. For instance, [15, 17] show that when order-preserving and deterministic encryption techniques are used together, on a dataset in which the entropy of the values is not high enough, an attacker might be able to construct the entire plaintext by doing a frequency analysis of the encrypted data. Third,

*__The approach sketched in this poster may be found in details in its ICDE 2019 conference version [16].__ This material is based on research sponsored by DARPA under agreement number FA8750-16-2-0021. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government. This work is partially supported by NSF grants 1527536 and 1545071.

CODASPY '19, March 25–27, 2019, Richardson, TX, USA

ACM ISBN 978-1-4503-6099-9/19/03.
https://doi.org/10.1145/3292006.3302384

	EId	FirstName	LastName	SSN	Office#	Department
t_1	E101	Adam	Smith	111	1	Defense
t_2	E259	John	Williams	222	2	Design
t_3	E199	Eve	Smith	333	2	Design
t_4	E259	John	Williams	222	6	Defense
t_5	E152	Clark	Cook	444	1	Defense
t_6	E254	David	Watts	555	4	Design
t_7	E159	Lisa	Ross	666	2	Defense
t_8	E152	Clark	Cook	444	3	Design

Figure 1: A relation: *Employee*.

mechanisms based on secret-sharing [20] are potentially more scalable; however, splitting data amongst multiple non-colluding cloud operators incurs significant communication overheads and can only support a limited set of selection and aggregation queries efficiently. Finally, SGX-based solutions also leak information during a query execution due to different attacks on SGX (*e.g.*, cache-line, branch shadowing, and page-fault attacks [12, 23]) and are significantly slower when overcoming these attacks using ORAM-based computations or emerging architectures such as T-SGX [21] or Sanctum [6].

While the race to develop cryptographic solutions that (*i*) are efficient, (*ii*) support complex SQL queries, (*iii*) offer provable security from the application's perspective is ongoing, this poster outlines a different (but complementary) approach to secure data processing by partitioning a computation over the public cloud based on the data classification into sensitive and non-sensitive data. We focus on an approach for situations when only part of the data is sensitive, while the remainder (that may consist of the majority) is non-sensitive. In particular, we consider a **partitioned computation model** that exploits such a classification of data into sensitive/non-sensitive subsets to develop efficient data processing solutions with **provable security guarantees**. Partitioned computing potentially provides significant benefits by (*i*) avoiding (expensive) cryptographic operations on non-sensitive data, and, (*ii*) allowing query processing on non-sensitive data to exploit indices.

2 PARTITIONED COMPUTATIONS

Let R be a relation that is partitioned into two sub-relations, $R_e \supseteq R_s$ and $R_p \subseteq R_{ns}$, such that $R = R_e \cup R_p$. The relation R_e contains all the sensitive tuples (denoted by R_s) of the relation R and will be stored in encrypted form in the cloud. The relation R_p refer to the sub-relation of R that will be stored in plaintext on the cloud. Naturally, R_p does not contain any sensitive tuples. For the remainder of the poster, we will assume that $R_e = R_s$ and $R_p = R_{ns}$. A partition computation strategy splits the execution of Q into two independent sub-queries: Q_s: a query to be executed on $E(R_e)$ and Q_{ns}: a query to be executed on R_{ns}. The final results are computed (using a query Q_{merge}) by appropriately merging the results of the two sub-queries at the trusted database (DB) owner side (or in the cloud, if a trusted component, *e.g.*, Intel SGX, is available for such a merge operation). In particular, the query Q on a relation R is partitioned, as follows: $Q(R) = Q_{merge}\Big(Q_s(R_s), Q_{ns}(R_{ns})\Big)$.

Let us illustrate partitioned computations through an example.

Example: Consider an *Employee* relation, see Figure 1. In this relation, the attribute *SSN* is sensitive, and furthermore, all tuples of employees for the *Department* = "`Defense`" are sensitive. In such a case, the *Employee* relation may be stored as the following three relations: (*i*) *Employee1* with attributes *EId* and *SSN* (see Figure 2a); (*ii*) *Employee2* with attributes *EId*, *FirstName*, *LastName*, *Office#*, and *Department*, where *Department* = "`Defense`" (see Figure 2b); and

EId	SSN
E101	111
E259	222
E199	333
E152	444
E254	555
E159	666

(a) A sensitive relation: *Employee1*.

	EId	FirstName	LastName	Office	Dept
t_1	E101	Adam	Smith	1	Defense
t_4	E259	John	Williams	6	Defense
t_5	E152	Clark	Cook	1	Defense
t_7	E159	Lisa	Ross	2	Defense

(b) A sensitive relation: *Employee2*.

	EId	FirstName	LastName	Office	Dept
t_2	E259	John	Williams	2	Design
t_3	E199	Eve	Smith	2	Design
t_6	E254	David	Watts	4	Design
t_8	E152	Clark	Cook	3	Design

(c) A non-sensitive relation: *Employee3*.

Figure 2: Three relations obtained from *Employee* relation.

(*iii*) *Employee3* with attributes *EId*, *FirstName*, *LastName*, *Office#*, and *Department*, where *Department* <> "`Defense`" (see Figure 2c). Since the relations *Employee1* and *Employee2* (Figures 2a and 2b) contain only sensitive data, these two relations are encrypted before outsourcing, while *Employee3* (Figure 2c), which contains only non-sensitive data, is outsourced in clear-text. We assume that the sensitive data is strongly encrypted such that the property of *ciphertext indistinguishability* (*i.e.*, an adversary cannot distinguish pairs of ciphertexts) is achieved. Thus, the two occurrences of `E152` have two different ciphertexts.

Consider a query `Q: SELECT FirstName, LastName, Office#, Department from Employee where FirstName = ''John''`. In partitioned computation, the query `Q` is partitioned into two sub-queries: Q_s that executes on `Employee2`, and Q_{ns} that executes on `Employee3`. Q_s will retrieve the tuple t_4 while Q_{ns} will retrieve the tuple t_2. Q_{merge} in this example is simply a union operator. Note that the execution of the query `Q` will also retrieve the same tuples.

Inference Attack in Partitioned Computations. Partitioned computations, if performed naively, could lead to inferences about sensitive data from non-sensitive data. To see this, consider following three queries on the *Employee2* and *Employee3* relations: (*i*) retrieve tuples of the employee `Eid = E259`, (*ii*) retrieve tuples of the employee `Eid = E101`, and (*iii*) retrieve tuples of the employee `Eid = E199`. We consider an *honest-but-curious* adversarial cloud that returns the correct answers to the queries but wishes to know information about the encrypted sensitive tables, *Employee1* and *Employee2*. Table 2 shows what does the adversary learn after executing the above three queries assuming that the tuple retrieving cryptographic approaches are not hiding access-patterns. During the execution, the adversary gains complete knowledge of non-sensitive tuples returned, and furthermore, knowledge about which encrypted tuples are returned as a result of Q_s ($E(t_i)$ in the table refers to the encrypted tuple t_i).

Query value	Returned tuples/Adversarial view	
	Employee2	Employee3
E259	$E(t_4)$	t_2
E101	$E(t_1)$	null
E199	null	t_3

Table 2: Queries and returned tuples.

Partitioned Data Security. In order to prevent such above-mentioned inference attack, we need a new security definition. Informally, partitioned data security requires us to ensure that: (*i*) The posterior probability that a sensitive record and a non-sensitive record are related (e.g., encode the same value) remains identical to the prior probability before query execution, and (*ii*) The posterior probability about the distribution (relative frequency) of sensitive values remains identical to the prior probability.

3 QUERY BINNING

Query binning (QB) is related to bucketization, which is studied in past [13]. While bucketization was carried over the data in [13], QB performs bucketization on queries. In general, one may ask more queries than original query while adding overhead but it prevents the above-mentioned inference attack. We provide QB under some assumptions and settings, given below.[1]

Problem Setup. We assume the following two entities in our model: (*i*) *A database (DB) owner*: who splits each relation R in the database having attributes R_s and R_{ns} containing all sensitive and non-sensitive tuples, respectively. (*ii*) *A public cloud*: The DB owner outsources the relation R_{ns} to a public cloud. The tuples in R_s are encrypted using any existing mechanism before outsourcing to the same public cloud. However, in the approach, we use non-deterministic encryption, *i.e.*, the cipher representation of two occurrences of an identical value has different representations.

DB Owner Assumptions. In our setting, the DB owner stores some (limited) metadata such as searchable values and their frequency counts, which are used for appropriate query formulation. The DB owner is assumed to have sufficient storage for such metadata, and also computational capabilities to perform encryption and decryption. The size of metadata is significantly smaller than the size of the original data.

Assumptions for QB. We develop QB initially under the assumption that queries are only on a single attribute, say A. The QB approach takes as inputs: (*i*) the set of data values (of the attribute A) that are sensitive along with their counts, and (*ii*) the set of data values (of the attribute A) that are non-sensitive, along with their counts. The QB returns a partition of attribute values that form the query bins for both the sensitive as well as for the non-sensitive parts of the query.

The Approach. We develop an efficient approach to execute selection queries securely (preventing the information leakage as shown above) by appropriately partitioning the query at a public cloud, where sensitive data is cryptographically secure while non-sensitive data stays in cleartext. For answering a selection query, naturally, we use any existing cryptographic technique on sensitive data and a simple search on the cleartext non-sensitive data.

Informally, QB distributes attribute values in a matrix, where rows are sensitive bins, and columns are non-sensitive bins. For example, suppose there are 16 values, say 0, 1, . . . , 15, and assume all the values have sensitive and associated non-sensitive tuples. Now, the DB owner arranges 16 values in a 4×4 matrix, as follows:

	NSB_0	NSB_1	NSB_2	NSB_3
SB_0	11	2	5	14
SB_1	10	3	8	7
SB_2	0	15	6	4
SB_3	13	1	12	9

In this example, we have four sensitive bins: SB_0 {11,2,5,14}, SB_1 {10,3,8,7}, SB_2 {0,15,6,4}, SB_3 {13,1,12,9}, and four non-sensitive bins: NSB_0 {11,10,0,13}, NSB_1 {2,3,15,1}, NSB_2 {5,8,6,12}, NSB_3 {14,7,4,9}. When a query arrives for a value, say 1, the DB owner searches for the tuples containing values 2,3,15,1 (viz. NSB_1) on the non-sensitive data and values in SB_3 (viz., 13,1,12,9) on the sensitive data using the cryptographic mechanism integrated into QB. While the adversary learns that the query corresponds to one of the four values in NSB_1, since query values in SB_3 are encrypted, the adversary does not learn any sensitive value or a non-sensitive value that is identical to a clear-text sensitive value.

[1]These assumptions are made primarily for ease of the exposition and relaxed in [16].

Based on QB, for answering the above-mentioned three queries, given in §2, QB creates two sets or bins on sensitive parts: sensitive bin 1, denoted by SB_1, contains {E101, E259}, sensitive bin 2, denoted by SB_2, contains {E152, E159}, and two sets/bins on non-sensitive parts: non-sensitive bin 1, denoted by NSB_1, contains {E259, E254}, non-sensitive bin 2, denoted by NSB_2, contains {E199, E152}. Table 3 shows that when answering a query using QB, the adversary cannot learn which employee works only in defense, design, or in both.

Query value	Returned tuples/Adversarial view	
	Employee1	Employee2
E259	$E(t_4), E(t_1)$	t_2, t_6
E101	$E(t_4), E(t_1)$	t_3, t_8
E199	$E(t_4), E(t_1)$	t_3, t_8

Table 3: Queries and returned tuples when following QB.

Evaluation. Table 4 shows the time taken when using QB with SGX-based Opaque and MPC-based Jana at different levels of sensitivity. Without using QB for answering a simple selection query, Opaque [24] took 89 seconds on a dataset of size 700MB (TPC-H LineItem table having 6M tuples) and Jana [3] took 1051 seconds on a dataset of size 116MB (1M tuples), while the time to execute the same query on cleartext data of size 700MB took only 0.0002 seconds.

Technique	1%	5%	20%	40%	60%
SGX-based Opaque [24]	11	15	26	42	59
MPC-based Jana [3]	22	80	270	505	749

Table 4: Time (in seconds) when mixing QB with Opaque and Jana at different levels of sensitivity.

REFERENCES

[1] Rakesh Agrawal et al. 2004. Order-Preserving Encryption for Numeric Data. In *SIGMOD*. 563–574.
[2] Arvind Arasu et al. 2013. Orthogonal Security with Cipherbase. In *CIDR*.
[3] David W. Archer et al. 2018. From Keys to Databases - Real-World Applications of Secure Multi-Party Computation. *Comput. J.* 61, 12 (2018), 1749–1771.
[4] Elette Boyle et al. 2015. Function Secret Sharing. In *EUROCRYPT*.
[5] Victor Costan et al. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* (2016).
[6] Victor Costan et al. 2016. Sanctum: Minimal Hardware Extensions for Strong Software Isolation. In *USENIX Security*. 857–874.
[7] Shlomi Dolev et al. 2015. Accumulating Automata and Cascaded Equations Automata for Communicationless Information Theoretically Secure Multi-Party Computation: Extended Abstract. In *SCC@ASIACCS*. 21–29.
[8] Shlomi Dolev et al. 2016. Private and Secure Secret Shared MapReduce. In *DBSec*. 151–160.
[9] Fatih Emekçi et al. 2014. Dividing secrets to secure data outsourcing. *Inf. Sci.* 263 (2014), 198–210.
[10] Craig Gentry. 2009. *A fully homomorphic encryption scheme.* Ph.D. Dissertation.
[11] Niv Gilboa et al. 2014. Distributed Point Functions and Their Applications. In *EUROCRYPT*. 640–658.
[12] Johannes Götzfried et al. 2017. Cache Attacks on Intel SGX. In *EUROSEC*. 2:1–2:6.
[13] Hakan Hacigümüs et al. 2002. Providing Database as a Service. In *SIGMOD*. 29–38.
[14] Yuval Ishai et al. 2016. Private Large-Scale Databases with Distributed Searchable Symmetric Encryption. In *RSA*. 90–107.
[15] Georgios Kellaris et al. 2016. Generic Attacks on Secure Outsourced Databases. In *CCS*. 1329–1340.
[16] Sharad Mehrotra et al. 2019. Partitioned Data Security on Outsourced Sensitive and Non-sensitive Data. In *ICDE*. Also as a technical report at Department of Computer Science, UC, Irvine, http://isg.ics.uci.edu/pubs/tr/partitioned.pdf.
[17] Muhammad Naveed et al. 2015. Inference Attacks on Property-Preserving Encrypted Databases. In *SIGSAC*. 644–655. https://doi.org/10.1145/2810103.2813651
[18] Rishabh Poddar et al. 2016. Arx: A Strongly Encrypted Database System. *IACR Cryptology ePrint Archive* (2016). http://eprint.iacr.org/2016/591
[19] Raluca A. Popa et al. 2012. CryptDB: processing queries on an encrypted database. *Commun. ACM* 55, 9 (2012), 103–111. https://doi.org/10.1145/2330667.2330691
[20] Adi Shamir. 1979. How to Share a Secret. *Commun. ACM* 22 (1979), 612–613.
[21] Ming-Wei Shih et al. 2017. T-SGX: Eradicating Controlled-Channel Attacks Against Enclave Programs. In *NDSS*.
[22] Dawn Xiaodong Song et al. 2000. Practical Techniques for Searches on Encrypted Data. In *IEEE SP*. 44–55. https://doi.org/10.1109/SECPRI.2000.848445
[23] Wenhao Wang et al. 2017. Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX. In *CCS*. 2421–2434.
[24] Wenting Zheng et al. 2017. Opaque: An Oblivious and Encrypted Distributed Analytics Platform. In *NSDI*. 283–298.