ISBN: 978-981-18-3959-7
2022 the 12th International Workshop on Computer Science and Engineering (WCSE 2022)
doi: 10.18178/wcse.2022.06.010

PENDA: Privacy ENhanced Data Aggregator

Jestine Paul [1,2 +], Saravanan Rajamanickam [1], Bharadwaj Veeravalli [2] and Khin Mi Mi Aung [1]

[1] Institute for Infocomm Research, A*STAR, Singapore 138632

[2] Department of Electrical and Computer Engineering, National University of Singapore, Singapore 117583

Abstract. Data is spread across different organizations and must be combined to get valuable analytics or train machine learning models. Sensitive features such as identification numbers, as columns, are common in these data, and organizations can only link if they share these columns. However, data protection regulations prevent these organizations from revealing the values in these columns to others. This paper proposes a technique, PENDA (Privacy ENhanced Data Aggregator), to encrypt columns of a database table or spreadsheet so that a central aggregation server can join them without decrypting using XOR-based homomorphism. We implement our PENDA system and demonstrate how organizations taking part in the process can encrypt and merge data. The experimental results show that the system can handle very large data files and scale to multiple organizations.

Keywords: encryption, cryptography, database

1. Introduction

Spreadsheets and relational databases make up most of the storage and processing of today's data. A spreadsheet is a file of cells in columns and rows, and it helps in sorting, arranging, and calculating data. A database collects information organized in a specific structure to quickly read, edit, add, or delete data. Codd [1] proposed the relational database model, tables with rows and columns representing items and their attributes. A special-purpose declarative language called structured query language (SQL) is used to query the relational database.

The inception of cloud computing permits anyone to outsource storage and computation on large quantities of data to third-party servers. However, consumers usually do not trust cloud service providers with confidential data [2]. This situation leads to privacy concerns, and threats from hostile insiders and external attackers fortify this lack of trust. They do not wish to endow third-party access to their private data, preventing database applications and spreadsheets from using cloud storage solutions. However, on-premise solutions are usually inconvenient than these large-scale ones and are also more susceptible to attacks. This situation demands a cryptographic solution that lets data on the cloud be end-to-end encrypted to make sure the server never reads the private plaintext data. However, this strategy causes a challenge when the server performs relational database operations on the data.

In distributed relational databases, data is stored across multiple sites, and some parts of this data are very private for its owners. This data distribution is a different scenario compared to a single centralized database server. Therefore, a natural cryptographic problem arises in this setting: how to generate helpful information while keeping all participating sites' data confidential? The most effective procedure still missing is a join operation based on secret-shared key columns. This operation can be used, e.g., to combine customer data from different organizations into a single dataset, as shown in Figure 1. The organization in the figure can be public and private sectors such as the tax department and a private bank.

Proposals to solve this problem include general-purpose primitives like fully homomorphic encryption (FHE) [3], functional encryption (FE) [4] and oblivious RAM (ORAM) [5]. FHE allows an untrusted server to receive encrypted data $enc(m)$ and perform any function f on it. It computes the encrypted result $enc(f(m))$, and $f(m)$ is only obtained by decrypting the result. FE allows a trusted authority holding a master secret key to generate a special functional secret key associated with a function f. The FE operation

+ Corresponding author.
E-mail address: jestine_paul@i2r.a-star.edu.sg

on $enc(m)$ with this key gives $f(m)$ without revealing anything about m. ORAM algorithms let a user hide its access pattern to the remote untrusted storage by shuffling and encrypting data again as they are accessed. Even though these techniques achieve such an "ideal" notion of privacy, they are unsuitable for real-life implementation due to significant performance and communication overheads. The problem of searching on encrypted data was introduced by Song et al. [6], where they presented a searchable symmetric key encryption scheme. This scheme allows a user to test if a ciphertext block contains a word, given a trapdoor for that word. We use a similar trapdoor approach to merge the tables.

The main contribution of this paper is to develop a mechanism where an organization can share private data required to combine data from other organizations. The proposed system handles database tables and spreadsheets consisting of millions of records, and it is also able to scale to merge data from multiple organizations.

The remainder of the paper is organized as follows. In section 2, there is a brief overview of related work and preliminary. The explanation of the proposed system and methodology are provided in section 3. Section 4 presents the system setup and experimental results with evaluation metrics. Finally, we conclude with a summary of our main contributions and results.

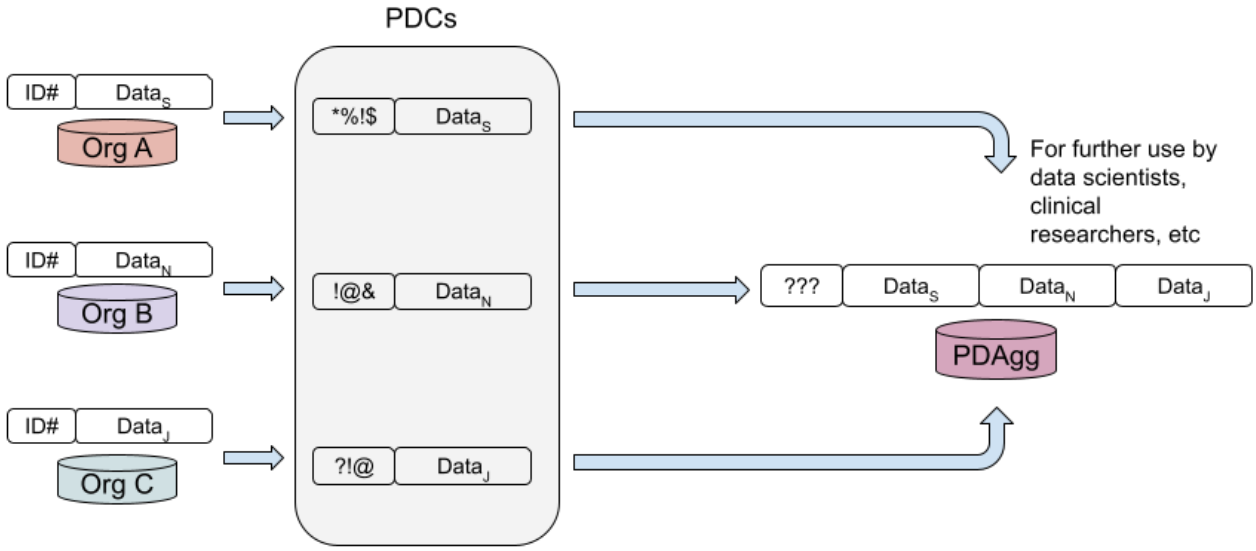

Fig. 1: Overall design of the proposed system

2. Preliminary

In this section, we will briefly review the searchable encryption scheme. We also go through the XOR homomorphism property, which we combine with searchable encryption in our system.

2.1. Searchable Encryption

The encrypted search was first considered explicitly by Song et al. in [6], which presented the concept of searchable symmetric encryption. A particular two-layered encryption construct is composed for all the words in the document. The queried keyword is converted to a trapdoor and searched by stripping its outer layer and checking whether the inner layer is of a specific format. The query returns only the document that finds a match and works when there is a low-bandwidth network connection between the client and the server.

Let us consider a scenario where Bob has a server where Alice wants to store her documents. However, she does not trust Bob. This situation could be an untrusted mail server where Alice, a mobile user, wants to keep her email messages. Therefore, Alice encrypts her documents with the searchable symmetric encryption scheme and only stores the ciphertext in the server. The documents' words can be considered tokens, such as a 64-bit block, and we pad the shorter words and split the longer words to obtain tokens of the same length.

To be more precise, Alice wants to encrypt a document containing a sequence of n bit words, $w_1, w_2 \dots w_q$. Instead of applying the process to plaintext $(w_1, w_2 \dots w_q)$, she encrypts w_i's as individual

blocks $x_i = E_{sk}(w_i)$. For each word x_i, she generates $n - m$ bit length string s_i using a stream cipher. After that, Alice split x_i into $L_i \,||\, R_i$ where L_i are $n - m$ bits (the same length as s_i) and R_i is m bits. Alice then uses a secret key k and a function G to create k_i s, i.e., $k_i = G_k(L_i)$. The trapdoor t_i is created by concatenating s_i and $F_{k_i}(s_i)$, i.e., $t_i = s_i \,||\, F_{k_i}(s_i)$ where F is a keyed function. The final ciphertext c_i is created by XOR operation of t_i with x_i, i.e., $c_i = x_i \oplus t_i$. To search for w_j, Alice gives the server (x_j, k_j). The server computes $t_i = c_i \oplus x_j$ for each i and checks if t_i is the form $s_i \,||\, F_{k_i}(s_i)$ to see any match.

2.2. XOR Homomorphism

A function f has XOR homomorphism property if on inputs x_1 and x_2, $f(x_1 \oplus x_2) = f(x_1) \oplus f(x_2)$. These functions preserve the structure of the XOR operation from the input to the output set. Examples of XOR homomorphism include bit-based circular shifts and permutations. If we apply n-bit left circular shifts on two strings and then XOR them, the result will be equal to the n-bit circular shift on the XOR of the original strings.

Secure permutation such as Fischer Yates shuffle generate uniformly distributed cycles of length n and has $O(n)$ runtime. Since all permutation has the XOR homomorphism property, these algorithms benefit secure search algorithms.

3. Our Approach

Encryption scheme such as Advanced Encryption Standard (AES) provides strong security against data theft but no other functionality. There is a considerable trade-off between privacy/security and utility. Hence, a variety of functionality-enhanced encryption is proposed, such as searchable encryption, order-preserving/revealing encryption, homomorphic encryption, and functional encryption. This section explains how we extend the searchable encryption for the secure merging of tables.

3.1. Encryption

Data stored in the cloud with traditional encryption does not allow searching. To search for data, we must download everything and decrypt it. Searchable encryption supports the creation of an encrypted database, where the data is tagged with encrypted keywords. The encrypted database can be uploaded to an external untrusted storage server and searched with encrypted queries. The server returns the data that is tagged by the encrypted query keyword.

PENDA encryption scheme uses similar techniques to search encryption to encrypt sensitive columns. Encrypt element E in the column using a deterministic encryption algorithm such as AES using a key k_1 to generate a 128-bit ciphertext.

$$W = AES_{k_1}(E)$$

Generate a random 64-bit string, X_L and shuffle it using a secure permutation function σ such as Fischer Yates shuffle into $\sigma(X_L)$. The shuffle generates the same permutation if the initial random number seed value is the same. Hence, the seed value is used as a key k_2. The final ciphertext is formed the by XOR operation of W and X_L concatenated with $\sigma_{k_2}(X_L)$ as shown in Ren et al. [7]

$$PENDA\_Encrypt(E) = W \oplus (X_L \,||\, \sigma_{k_2}(X_L))$$

This encryption, shown in Figure 2, is non-deterministic since the same plaintext can generate multiple ciphertexts depending on the random number X_L generated.

3.2. Indexing

Consider we have two ciphertexts A and B, which we want to test if they are equal. If they are equal, the deterministic AES encryption output W will be the same for both A and B. Then A and B will be of the following form

$$A = W \oplus (XA_L \,||\, XA_R)$$
$$B = W \oplus (XB_L \,||\, XB_R)$$
$$\text{where } XA_R = \sigma(XA_L) \text{ and } XB_R = \sigma(XB_L)$$

To check for equality, we first compute the XOR operation of A and B.

$$A \oplus B = (W \oplus (XA_L \,||\, \sigma(XA_L))) \oplus (W \oplus (XB_L \,||\, \sigma(XB_L))$$

This will eliminate the common W component of the ciphertexts. Hence

$$A \oplus B = (XA_L \oplus XB_L) \,||\, (\sigma(XA_L) \oplus \sigma(XB_L))$$

Fig. 2: PENDA encryption and unique tag creation

Next, we compute the secure permutation using Fisher-Yates shuffle on the left half of the $A \oplus B$ and check if it is equal to the right half. If the A and B are equal, the secure permutation preserves the XOR structure due to the XOR homomorphism property as shown below.

$$\sigma\,(XA_L \oplus XB_L) = \sigma(XA_L) \oplus \sigma(XB_L)$$

The merge operation will be costly if we perform the above equality test one pair at a time. It takes $O(n^2)$ comparison to combine two tables, and hence to compare k tables, it takes $O(n^2k)$ comparisons. Relational databases can be joined efficiently, creating an index on the column it is joined. The joining column can be bucketed with hash to create a hash table index. We take a similar approach to create an index, but the column has different ciphertexts for the same plaintext value. Hence, we outline a method to create a unique tag that corresponds to the same plaintext as shown in Figure 2.

We apply the secure permutation to the left half of the ciphertext. We then apply the XOR operation to the output of the permutation with the right half of the ciphertext. Since the permutation property preserves the XOR structure due to XOR homomorphism, the common permutation of the random string $\sigma(X)$ gets eliminated. This produces the output $\sigma(W_L) \oplus W_R$ which corresponds uniquely to the plaintext. This unique tag can then be used to create the index for the merge operation.

3.3. Merging

After the index is created using the hash of the tag, the tables can be merged by looking up the tag of each row. The complexity of this operation depends on the data structure used for storing the index. If a hash-table is used, then each lookup takes $O(1)$ steps and hence the merging of two tables takes $O(n)$ comparisons.

4. System Evaluation

4.1. Experimental Setup

We implemented the encryption and merging using Python. The AES encryption is implemented using the PyCryptodome[1] library. All the operations are scaled to use all the available threads in the CPU using the Dask [8] framework. We split the implementation into two software components: PDC (Privacy-Preserving Data Collector) and PDAgg (Privacy-Preserving Data Aggregator). PDC does the PENDA encryption, and PDAgg does the PENDA indexing and merging.

Each organization that wants to participate in the process encrypts its data using the PDC software. We assume that the AES key and the key for secure permutation are shared with the organizations using a secure

[1] https://www.pycryptodome.org/

key distribution protocol. The PDA software is used to encrypt personally identifiable information such as ID numbers. Since the PENDA encryption is a non-deterministic scheme, the same ID number will produce different ciphertexts in different organizations, as shown in Figure 3. The AES key is shown in red, and the key for secure permutation is shown in yellow.

Once all the encrypted files are securely transferred to the PDAgg, it first proceeds with index creation. The unique tag is generated for each row using the PENDA indexing algorithm, as shown in Figure 4. As before, we assume the key for secure permutation is shared with the PDAgg along with PDC using a secure key distribution protocol. Once the unique tag is generated, the index is created and merged, as shown in Figure 5.

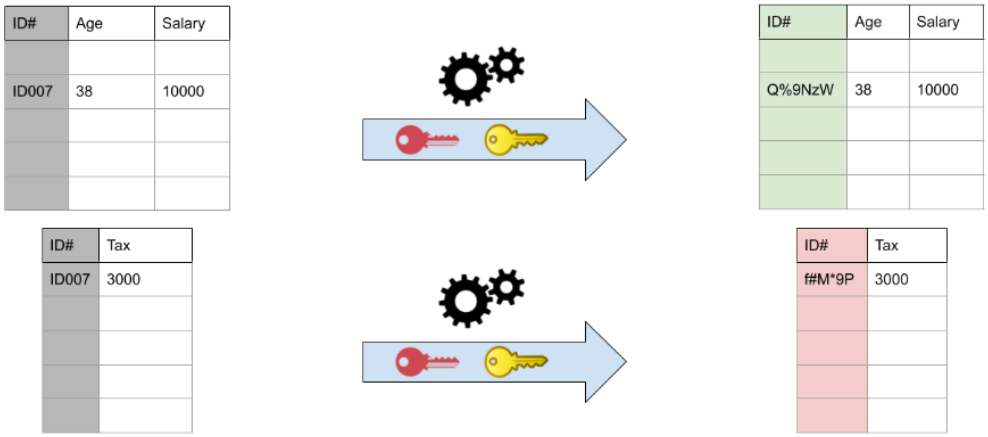

Fig. 3: Encrypting ID# column where different ciphertexts are produced for the same plaintext.

4.2. Experimental Results

In our first experiment, we simulated three organizations with one million records on a machine with an Intel Xeon Platinum 8170 processor running Debian Linux 11. The PDC and PDAgg were run utilizing all threads in the CPU. All the threads were profiled with Python yappi[2] profiler, and the wall time recorded is shown in Table 1.

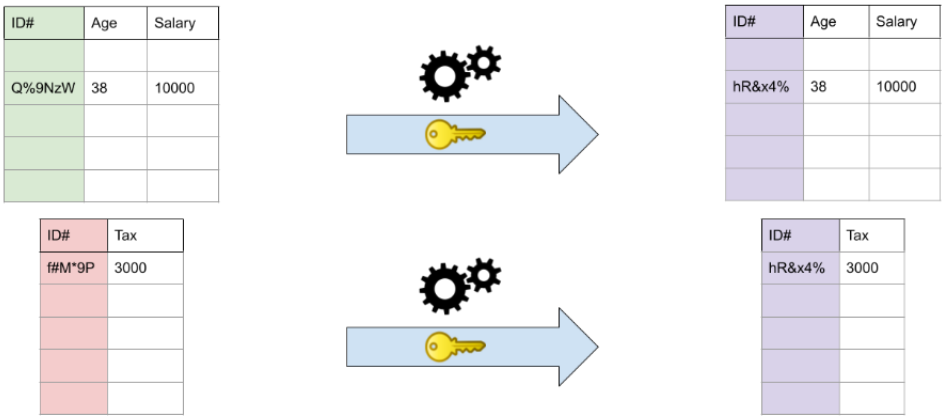

Fig. 4: Generating the unique tag from ID# ciphertexts

For the case of 5 million rows, the PDC could encrypt up to 200,000 rows per second. The PDAgg took 36.87 seconds to generate the unique tag for each file, and hence to merge the three files takes $36.87 \times 3 + 95.75 = 206.3$ seconds, assuming that the unique tag generation is done in sequence. We simulated up to 10 organizations in our next experiment. In the first case, each organization had 500,000 records, and we repeated the experiment with one and two million records. The total aggregation time at PDAgg is shown in Figure 6. We assume that the unique tag generation is done in sequence.

[2] https://github.com/sumerc/yappi

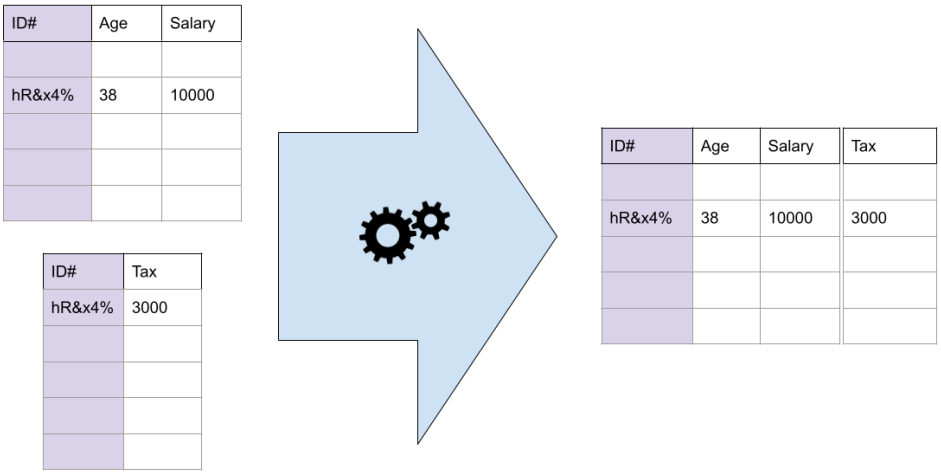

Fig. 5: The tables merged using the unique tag.

These experiments show that the merging using the PENDA algorithm is faster than other techniques using FHE, FE or ORAM. A simple equality comparison on ciphertext encrypted with FHE [9] takes an order of seconds.

Table 1: Time taken for each operation for three organizations

Rows	Encryption Time (PDC)	Unique Tag Generation Time (PDAgg)	Merging Time (PDAgg)
100,000	1.98s	1.52s	0.60s
500,000	3.69s	4.41s	3.16s
1,000,000	6.24s	7.89s	6.59s
2,000,000	10.65s	15.29s	36.76s
5,000,000	25.20s	36.87s	95.75s

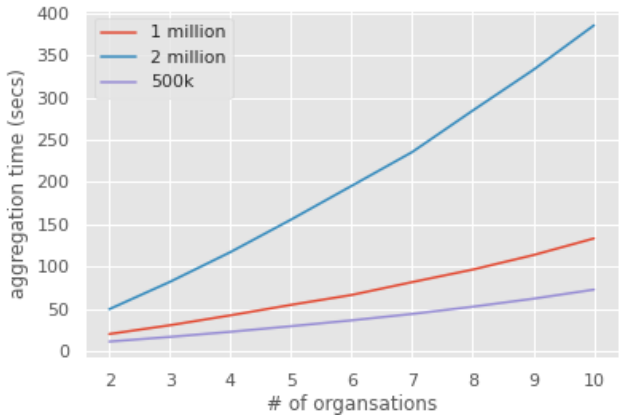

Fig. 6: Execution time of PDAgg versus the number of organisations.

5. Conclusion

In this paper, we have proposed an efficient scheme PENDA that encrypts columns of a database table or spreadsheet so that a central aggregation server can join them without decrypting using XOR-based homomorphism. Data is locked in siloes in different organizations due to privacy concerns, and it is a vast untapped resource for data mining. Using our proposed method, we merge data across these organizations without revealing personally identifiable information such as ID numbers. The algorithm was implemented and executed on files with millions of rows. It may be noted that the focus of this proposed encryption

scheme is only on the shared column on which it merges. Hence, an immediate extension of this work is to consider how to encrypt other unshared columns homomorphically to be used for computation without decrypting.

6. Acknowledgment

This research is supported by Institute for Infocomm Research, A*STAR Research Entities under its RIE2020 Advanced Manufacturing and Engineering (AME) Programmatic Programme (Award A19E3b0099).

7. References

[1] Codd, E. (1983). A Relational Model of Data for Large Shared Data Banks. *Commun. ACM, 26(1), 64–69.*

[2] Chu, C.K., Zhu, W.T., Han, J., Liu, J., Xu, J., & Zhou, J. (2013). Security Concerns in Popular Cloud Storage Services. *IEEE Pervasive Computing, 12(4), 50-57.*

[3] Gentry, C. (2009). Fully Homomorphic Encryption Using Ideal Lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing* (pp. 169–178). Association for Computing Machinery.

[4] Boneh, D., Sahai, A., & Waters, B. (2012). Functional Encryption: A New Vision for Public-Key Cryptography. *Commun. ACM, 55(11), 56–64.*

[5] Goldreich, O., & Ostrovsky, R. (1996). Software Protection and Simulation on Oblivious RAMs. *J. ACM, 43(3), 431–473.*

[6] Song, D., Wagner, D., & Perrig, A. (2000). Practical techniques for searches on encrypted data. In *Proceeding 2000 IEEE symposium on security and privacy. S&P 2000* (pp. 44–55).

[7] Ren, S., Tan, B., Sundaram, S., Wang, T., Ng, Y., Chang, V., & Aung, K. (2016). Secure searching on cloud storage enhanced by homomorphic indexing. *Future Generation Computer Systems, 65, 102–110.*

[8] Rocklin, M. (2015). Dask: Parallel computation with blocked algorithms and task scheduling. In *Proceedings of the 14th python in science conference*.

[9] Kim, M., Lee, H., Ling, S., Ren, S., Tan, B., & Wang, H. (2019). Search Condition-Hiding Query Evaluation on Encrypted Databases. *IEEE Access, 7, 161283-161295.*