Memory Efficient Data-Protection for Database Utilizing Secure/Unsecured Area of Intel SGX

Masashi Yoshimura
Division of Information Science
Nara Institute of Science and Technology
Nara, Japan
email: yoshimura.masashi.yj6@is.naist.jp

Taisho Sasada
Division of Information Science
Nara Institute of Science and Technology
Research Fellow of the JSPS
Nara, Japan
email: sasada.taisho.su0@is.naist.jp

Yuzo Taenaka
Division of Information Science
Nara Institute of Science and Technology
Nara, Japan
email: yuzo@is.naist.jp

Youki Kadobayashi
Division of Information Science
Nara Institute of Science and Technology
Nara, Japan
email: youki-k@is.naist.jp

***Abstract*—With the spread of cloud computing, database services have been provided on cloud platforms. As a Cloud Service Provider (CSP) has the highest privilege in the cloud platform, the CSP can get any data from the database even if a tenant admin secures all components, such as OS, database software, etc. as long as the database runs on the cloud. That is why CSP has become a new threat source in cloud-based databases. Trusted Execution Environment (TEE) is a key technology to protect memory, process, and storage against data theft by a CSP. It creates a secure area on the memory where the process outside the secure area cannot access, thereby preventing any access from CSP. However, since the secure area only has a limited amount of memory resources on a server, the rest memory resources keep vacant even when TEE exhaustively uses its allocated memory resources. In the case of the high-load database running on the secure area, almost all queries slow down due to being full of consumed memory despite most of the memory being free in the unsecured area. In this study, we design an efficient memory management mechanism for TEE-based secure database that effectively uses the resources of both the secure and unsecured areas; the proposed system handles only sensitive queries and data in the secure area while others in the unsecured area. Experimental results show that our system improves both resource utilization efficiency and execution speed compared to the system processing all data in the secure area.**

***Index Terms*—Data Protection; RDBMS; Intel SGX; Trusted Execution Environment; Cloud Computing.**

I. Introduction

Along with the widespread of using cloud platforms, most services come to running on a cloud platform. Although the cloud is very useful for flexible service management adapting to time-varied workloads, it creates new threats to cybersecurity. A cloud platform runs tenant processes on the top of the virtualization layer, such as a hypervisor, and thus all processes, memory, and storage are accessible from the virtualization layer. That is, even if tenant admins strictly secure their OS, service processes, and data, CSP is able to affect processes, obtain data from storage or memory, etc.

One of the major and important systems running on the cloud is a relational database management system (RDBMS), which basically handles important data on the business. As most business is driven by data these days, protecting data is extremely essential for a database. For this purpose, most RDBMS, such as MySQL or PostgreSQL, has an encryption function that encrypts data on storage and protects against data theft. Nevertheless, as mentioned before, CSP can compromise even such encryption by taking process/memory from the virtualization layer. Therefore, it is necessary to protect data on RDBMS even on cloud systems consistently.

Existing studies provide a Trusted Execution Environment (TEE)-based solution to protect the data of the database [1] [2]. TEE creates a secure area where user programs are decrypted and executed directly on a CPU. As any process outside the secure area cannot access process/memory in the secure area, a database working in the secure area can protect its data in any case. Although TEE protects data even on the cloud, TEE has a limitation on the secure area; the secure area can only use quite less memory than what hardware has to run the program. That is why the performance of a high-load database hits the ceiling even if its physical hardware has more memory and most of it is still vacant. Studies [1] [3] run a database in the secure area and face the problem, while a study [2] migrates most of the process onto the unsecured area but increasing communication between secure and unsecured area, and finally these interactions become a performance bottleneck.

To overcome this problem, we propose a system that protects data in RDBMSs that extends the upper limit of TEE-based database performance. The main idea is to divide the whole processes of a database into two: a series of processes for sensitive data and a series of processes for other non-sensitive data. The proposed system allows the user to define the confidentiality of each column when creating the table and executes the former process on the secure area while the latter on the unsecured area, separately. The proposed system also reduces

 ISBN: 978-1-68558-056-8

the number of communication between the secure/unsecured areas because each data is handled in the secure or unsecured area all the time. The basic design of the RDBMS is based on open-source Postgresql [3]. We implement our proposal using Intel Software Guard Extensions (SGX), a hardware-based TEE of Intel CPU. For performance evaluation, we compare our proposal with the system processing all data in the secure area. We confirmed that the proposed method reduces secure area usage by 44% compared to existing methods and runs over 3.3× faster than existing methods when dealing with a large amount of data.

The structure of this paper is as follows: In Section 2, we explain the related works of databases using TEE. In Section 3, we describe the preliminary of Intel SGX. In Section 4, we explain the sensitive information and the design of our proposed system. In Section 5, we show the result of the experimental evaluation. In Section 6, we discuss the limitation of our proposed system and security vulnerabilities and in Section 7, we conclude our contribution.

II. Related Work

There are several TEE-based solutions to protect the data of RDBMS as related work [1]–[3]. EnclaveDB [1] is a system that makes Hekaton, an in-memory database engine included in Microsoft SQL Server, available as an SGX application. EnclaveDB ensures data confidentiality but handles all types of queries in the secure area. Therefore, the unsecured area has much vacant memory. CryptSQLite [3] proposed a system that ensures the data confidentiality and integrity of SQLite by storing all data in a secure area. That is the available memory in the unsecured area remains free. StealthDB [2] executes most of the database processes in the unsecured area while few sensitive processes are in the secure area. Although it can use the memory of both secure and unsecured areas, processes on the secure and unsecured areas require many interactions to handle a series of query processing. Encryption/decryption of data is necessary to protect data from going back and forth between secure and unsecured areas. Although this design contributes to reducing the load on the secure area, the interaction and the encryption/decryption are a very huge burden for the database. This results in a large overhead for even a simple SELECT statement that traverses a large amount of data. The proposed method divides database processes for the secure or unsecured area, similar to EnclaveDB or StealthDB, but makes these processes independent so as to avoid communication between secure and unsecured areas as much as possible. From this design, we realize the efficient use of memory in the secure area as well as the avoidance of performance bottlenecks that happened in the communication between the secure and unsecured area.

III. Intel SGX

Intel SGX utilizes the cryptographic engines in Intel CPU to create an isolated environment (secure area) called Enclave. We store data in Enclave, protecting program execution with guaranteed confidentiality and integrity. As Enclave does not provide storage, Intel SGX provides a function called Sealing/Unsealing. This function encrypts data using a key stored in the CPU; nobody except the CPU decrypts it. Moreover, for integrity assurance, Intel SGX provides a verification mechanism called Remote Attestation (RA), which can verify the integrity of programs within Enclave and the remote SGX platform. Thus, a client communicating with a remote SGX platform can send and receive data securely to and from CSPs using TLS sessions generated by RA.

Although Intel SGX provides a useful mechanism for protecting processes and memory, data, the mechanism inevitably includes several performance overheads. First, there is an Enclave size limitation for each Intel CPU version. For example, the 6th to 10th-generation Intel CPUs have a size limit of 128 MB, and the 3rd-generation Xeon scalable processors have a maximum size limit of 512 GB. Second, Intel SGX supports Enclave paging, but page swapping incurs an overhead of about 40,000 CPU cycles due to page copy and context switches and so on [4]. Therefore, when we try to use many Enclave areas, much overhead is incurred. Third, SGX applications provide a transition between the Enclave process and the unsecured process. The transition function from an unsecured process to an Enclave process is called Ecall and the reverse transition function is called Ocall. However, during Ocall/Ecall, SGX performs context switches and flushes Translation Lookaside Buffer, resulting in an overhead of about 8,000 to 17,000 CPU cycles [5].

IV. Proposed Method

Before going into the detail of our proposal, we describe the threat model. Our threat model is information leakage at a database server. The adversary is a malicious CSP that has free access to memory and storage. Specifically, the adversary can steal data in memory by memory dump or cold boot attacks and data in storage by physically obtaining storage devices. Note that the retrieval of sensitive data or encryption keys directly from Intel SGX [6] [7] is beyond the scope.

As a general database that supports many kinds of queries, this paper limits the target queries for simplicity to basic CRUD operations (CREATE, INSERT, SELECT, UPDATE, and DELETE). Note that the support of queries such as subqueries and joins is future work.

For making processes independent for the secure and unsecured area, we design a table with secure columns. We focused on the fact that database tables generally consist of several columns having either sensitive data or non-sensitive data. For example, suppose that a table in a database that stores company employee information includes name, age, and hometown. In this case, only the hometown is non-sensitive information because of public data while name and age are sensitive. As these sensitive or non-sensitive data are different in a table, the proposed system allows users to define the confidentiality of data on a column-by-column basis when the table is created. In the proposed system, only the processing of sensitive data is performed in the secure area, and all other processing is performed in the unsecured area in order

 ISBN: 978-1-68558-056-8

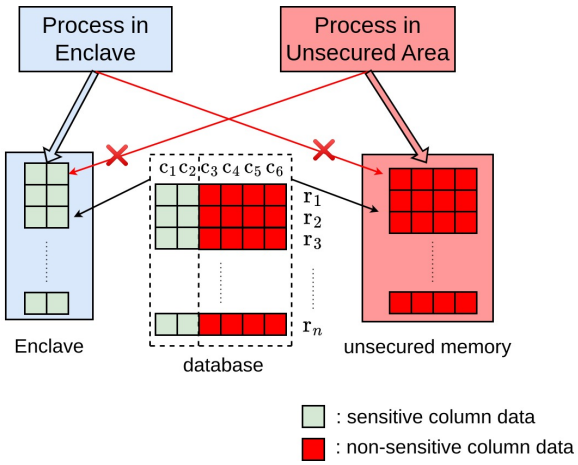

Figure 1. Proposed System Overview.

to improve resource utilization efficiency. Figure 1 shows this concept where columns of sensitive data, called sensitive columns, (c1 and c2) are treated in the secure area while columns of non-sensitive data, called non-sensitive columns, are processed in the unsecured area.

A. Sensitive Information

The proposed architecture is shown in Figure 2. We handle four types of information that may contain sensitive data in the proposed architecture: (1) queries, (2) query trees, (3) plan trees, and (4) sensitive buffer pools. We describe them one by one.

(1) Query: As queries may contain sensitive data (e.g., INSERT statements including sensitive data or queries containing WHERE clauses that specify values of sensitive data), all queries must be processed in the secure area. If the query does not contain sensitive data, data processing for the query is done in the unsecured area.

(2) Query Tree: The query tree is an abstract syntax tree of a query and therefore contains the same sensitive information as the query. Therefore, our system processes the query tree in the secure area.

(3) Plan Tree: A plan tree is a tree structure that shows the optimal query plan for a query tree. As a plan tree is used to issue instructions to process data in reality, it must be separated for the secure and unsecured areas in accordance with sensitive or non-sensitive columns. To avoid communication between two areas, we here have to make two different (non-related) plan trees for secure and unsecured areas.

(4) Buffer Pool: Since each record contains both non-sensitive and sensitive data, the non-sensitive data of all records is deployed in the non-sensitive buffer pool in the unsecured area and the sensitive data of all records is deployed in the sensitive buffer pool in the secure area. Each buffer pool is a fixed-length array of pages with a specified size (8KB, the same as the default setting of PostgreSQL), as in general RDBMS.

B. Design details of each module

The proposed system consists of 13 modules. We describe the function and key points of each module in the proposed method.

Communication Process, Decryption: The Communication Process (① in Figure 2) performs RA and query reception. A database client performs RA verification to determine whether or not the cloud server's platform (CPU) and the secure area can be trusted. If the client accepts the RA verification results and trusts the server platform and the secure area, the client encrypts the query using the symmetric key generated in the RA process and sends it to the cloud. After the communication process receives the encrypted query, it is sent to the secure area, and Decryption (② in Figure 2) decrypts the query using the symmetric key held within the secure area.

Parser: Parser (③ in Figure 2) generates a query tree from the query in the secure area.

Query Planner: The query planner (④ in Figure 2) generates a tree structure data called a plan tree representing the optimal query plan. The query planner optimizes a query tree so as to process the query efficiently, reducing the number of computations or data access to storage. The optimization is done for the entire query tree without taking care of non-sensitive or sensitive columns in this module.

Query Separator: The query separator (⑤ in Figure 2) can divide the optimal plan tree into a sensitive plan tree that handles only sensitive columns and a non-sensitive plan tree that handles only non-sensitive columns. If a query tree includes one or more sensitive columns, the query separator creates a new plan tree, called a sensitive plan tree, including sensitive columns only, which has the same structure as the original plan tree. Regarding non-sensitive columns, it makes the same tree remain non-sensitive columns only, which is called a non-sensitive plan tree. However, there are some queries that cannot be simply divided. Figure 3 shows the division of the plan tree of `SELECT (name, club) FROM USER WHERE country = 'Japan';`. The table USER consists of three columns: `name`, `club`, `country`. Only `name` is a sensitive column in this case. In this query, the `name` column data to be selected depends on `WHERE country = 'Japan'`, but the `country` column cannot be included in the sensitive plan tree because it is a non-sensitive column. In the proposed system, the sensitive plan tree generated by the division has an empty WHERE clause. After finishing the process of the non-sensitive plan tree, the WHERE clause of the sensitive plan tree refers to the identifiers ($id_1, id_2, ..., id_n$ in Figure 3) of records satisfying `WHERE country = 'Japan'`. Division of the plan tree is particularly important in the processing of complex queries with multiple subqueries, and it is the future work to design algorithms for processing like that queries efficiently utilizing the secure areas and reducing the number of communications between the secure and unsecured areas.

Query Executor: The Query Executor generates specific data processing instructions according to a plan tree. As the query separator makes two plan trees, non-sensitive and sensitive plan trees, the query executor is also required to be two in order to execute these two trees in the secure and unsecured area,

 ISBN: 978-1-68558-056-8

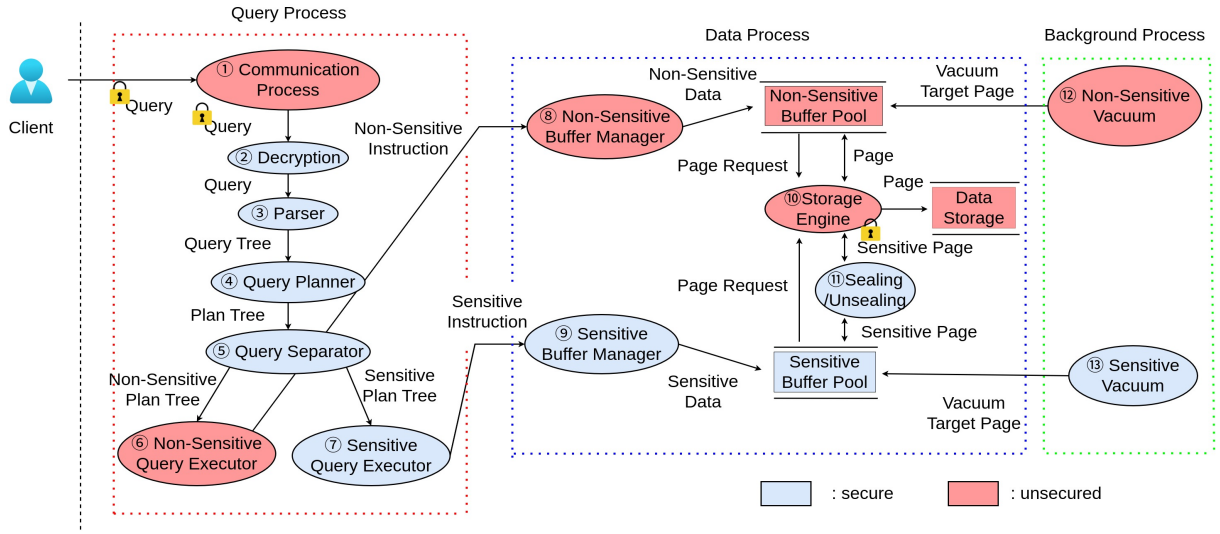

Figure 2. Architecture of the Proposed System.

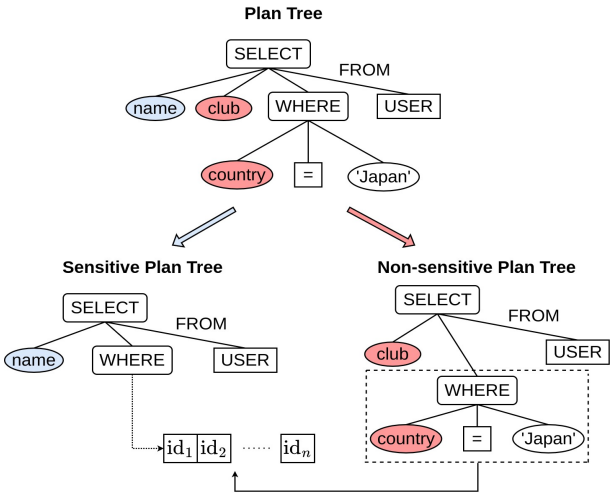

Figure 3. Division of the Plan Tree.

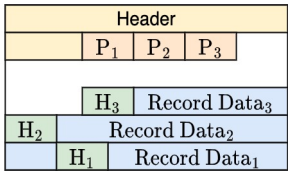

Figure 4. Data Page Structure in PostgreSQL.

respectively. We call it running in the secure area as a sensitive query executor while it is in the unsecured area as a non-sensitive query executor. Their function of them is identical except for the area they are running.

Buffer Manager: As a query is processed in parallel in both the secure and unsecured area, two buffer managers work in those areas, respectively, called sensitive and non-sensitive buffer managers. The functionality of the buffer manager is to handle the buffer pool based on data processing instructions issued by the Query Executor. The buffer manager in the unsecured area (⑧ in Figure 2) operates the non-sensitive buffer pool, while the buffer manager in the secure area (⑨ in Figure 2) operates the sensitive buffer pool.

Storage Engine: Storage Engine (⑩ in Figure 2) performs general file I/O processing and store pages of buffer pool in the storage. Storage engine stores sensitive buffer pools and non-sensitive buffer pools in storage for persistent data. To protect the sensitive buffer pool, the sensitive data in the buffer pool needs to be encrypted before going to the unsecured area. The encryption is done by using the sealing (⑪ in Figure 2) function of Intel SGX, which is the encryption function using a secret key of the CPU. To extract the encrypted data in the secure area, unsealing (⑪ in Figure 2) function is provided.

We explain the page structure of the buffer pool and how pages of the buffer pool are stored in the storage. The page structure of the buffer pool is similar to that of PostgreSQL as shown in Figure 4. PostgreSQL holds fixed-size memory for every page and manages data with a record unit on a page. On a page, every record is placed from the end of the page back-to-back, and those pointers (P) indicating the location of every record are put one by one after the header information. Note that a page contains the page header (Header in Figure 4) and the record header (H_1, H_2, H_3 in Figure 4). The page structure of the proposed system is the same as PostgreSQL but the page of the sensitive buffer pool has only sensitive data of records and the page of the non-sensitive buffer pool has only non-sensitive data of records. When storing buffer pool pages, sensitive data must be encrypted by Intel SGX sealing before storing sensitive pages. The simplest method is to encrypt the entire sensitive page. However, since the length of the encrypted byte array for the entire page exceeds 8KiB (page size), it becomes impossible to manage the data by fixed-length pages in the storage. In fact, it is sufficient to encrypt only the sensitive data on the sensitive page. Thus, a method to encrypt only the sensitive data of each record is considered, but it

 ISBN: 978-1-68558-056-8

TABLE I
STUDENT TABLE FOR EVALUATION

columns	id	name	university	club
type	integer	char(100)	char(100)	char(50)
attribution	normal	sensitive	normal	sensitive

```
// Query-1
INSERT INTO STUDENT (id, name, university, club)
    values (1, "John", "NAIST", "soccer");
// Query-2
UPDATE STUDENT set name = "Mike" where id = 1;
// Query-3
DELETE from STUDENT where id = 1;
// Query-4
SELECT * FROM STUDENT;
```

Figure 5. Transaction for Evaluation.

requires encryption of the number of records on a page, thereby incurring extra overhead. Considering the above, the proposed system adopts the method of encrypting all records (including the header of each record) on a sensitive page at once. In this way, only one encryption per page is required.

Vacuum Process: Vacuum Process is a background process that periodically cleans up buffer pools becoming dirty as a result of repeated data processing. This process runs in the secure area and unsecured areas (⑫, ⑬ in Figure 2) to handle sensitive and non-sensitive buffer pools, respectively.

V. EVALUATION

As a security evaluation, it is necessary to show that the confidentiality of sensitive information is ensured. In this study, the confidentiality of sensitive information means that the sensitive information exists as plain text only in the secure area and is always encrypted in the unsecured area. In Section IV, we can see that all sensitive information is processed in the secure area and is always encrypted before being sent to the unsecured area, so confidentiality is ensured.

We evaluated the performance of the proposed system. The experimental environment used for performance evaluation was Ubuntu 20.04LTS OS, on Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz, 4 CPU cores, SODIMM DDR3-1600 8GiB memory, Samsung SSD 860 500GiB. In this experimental environment, the available secure area is limited to 128 MiB at the same time due to the Intel CPU version. The proposed system was implemented with C++ and Intel SGXSDK [8], a development tool for SGX applications. In the performance evaluation, we evaluated the secure area usage (the amount of peak stack and heap memory in the secure area) and the execution time (a period between receiving a query from a client and generating a reply to the client) for the transaction of Figure 5. Note that the value of each query and the right-hand value of the WHERE clause vary by transaction.

We compare the secure area usage and execution time for processing 1, 10, 100, and 1000 transactions respectively on the proposed system and a comparative system (same features as EnclaveDB), which processes all data in the secure area. Also,

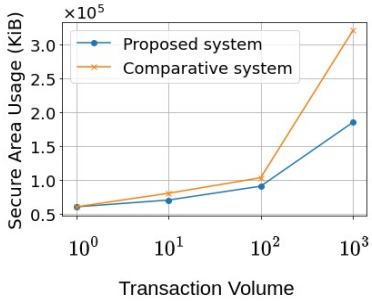

Figure 6. Peak Secure Stack and Heap Usage of Transactions.

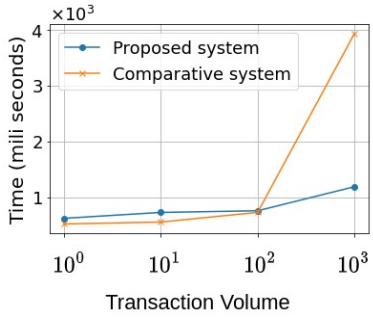

Figure 7. Execution Time of Transactions.

we ran the system three times for each transaction volume and used the average values of secure area usage and execution time as the evaluation values. The evaluation of secure area usage is shown in Figure 6, and the evaluation of execution time is shown in Figure 7. As Figure 6 shows, the proposed system uses less secure area than the comparative system for all 1, 10, 100, and 1000 transactions. As Figure 7 shows, the execution times of the proposed method and the comparative system are almost the same for 1, 10, and 100 transactions, but the overhead of the comparative system is about 3.3 times larger than that of the proposed method for 1000 transactions. Since the secure area available in the experimental environment is 128 MiB (96 MiB excluding reserved area), paging occurs very frequently for 1000 transactions of the comparative system, which uses much more than 96 MiB of the secure area. Thus, the proposed method improves execution speed over the comparative system when handling a large amount of sensitive data. In this evaluation, we used our own tables and transactions with transaction volumes of 10^0, 10^1, 10^2, and 10^3. It is future work to evaluate the results for larger transaction volumes and standard benchmark tests.

We explain the amount of change in secure areas and the number of communications between secure/unsecured areas as the number of tables, columns, and records increases. Since the database tables are managed in the unsecured area, the usage of the secured area does not increase even if the number of tables increases. When the number of sensitive columns or the number of records containing sensitive data increases, the usage of the secure area increases by the size of the secure area, but the number of communications between the secure/unsecured areas during query processing remains the same. However, if

 ISBN: 978-1-68558-056-8

TABLE II
SCORE TABLE

columns	id	name	score1	score2
type	integer	char(100)	integer	integer
attribution	non-sensitive	sensitive	non-sensitive	sensitive

TABLE III
SCORE DATABASE

id	name	score1	score2
1	John	86	68
2	Tom	95	98
3	Mike	58	99

the number of sensitive columns or records greatly increases, the secure area will be overutilized, and resource utilization efficiency will decrease. Thus, the proposed system can process even a large database consisting of multiple tables without incurring significant overhead if there is not a large amount of sensitive data.

VI. DISCUSSION

A. Extended RDBMS functionality

The current design ensures the confidentiality of sensitive data for queries that perform basic CRUD operations, but there are some queries that leak sensitive data. For example, suppose there is a table SCORE such as TABLE II and a database such as a TABLE III, and the query of Figure 8 is processed. In this query, Since `score1` is a non-sensitive column, `WHERE score1 < ···` is performed in the unsecured area. Thus, the return value of `SELECT score2 FROM WHERE id = 2` must be used in the unsecured area, leading to the leakage of sensitive data because `score2` is a sensitive column. However, since processes in the secure area can directly handle data in the unsecured area, we can compare sensitive data with non-sensitive data without storing the non-sensitive data in the secure area. It is necessary to process and evaluate such queries.

The system proposed in this study lacks important functions such as a transaction manager and a log manager, which are included in many RDBMS. It is necessary to evaluate whether or not the addition of such functions will ensure the confidentiality of sensitive data and whether or not the system can demonstrate practical performance.

B. Reduce Overhead due to communication between secure and unsecured area and Paging Reduction

As explained in Section III, communication between secure/unsecured areas and page swapping due to excessive use of secure areas incurs a large overhead. In the proposed system, these overheads are a serious problem when processing large amounts of sensitive data. As solutions to these problems, Intel SGX provides Switchless Call [9], which enables the communication between secure/unsecured areas without context switches and Eleos [10] enables paging within the secure area, thus reducing the paging overhead. The implementation of these techniques in our proposed system can improve performance.

```
SELECT * FROM SCORE
    WHERE score1 < (
        SELECT score2 FROM SCORE
            WHERE id = 2
    );
```

Figure 8. Sensitve Data Leakage Query.

C. Security Vulnerabilities

The secure area of Intel SGX is vulnerable to side-channel attacks, which can leak secret keys and internal registers [6] [7] [11] [12], but methods to mitigate these attacks significantly with little overhead are being studied [13] [14].

VII. CONCLUSION

In this paper, we proposed data protection for RDBMS that ensures data confidentiality while improving overall resource utilization efficiency using Intel SGX, a TEE with high-security features. The system efficiently utilizes the secure area by offloading only sensitive data and the processes that handle them to the secure area. In particular, in the case of handling both sensitive data and large amounts of non-sensitive data, the proposed system has improved both resource utilization efficiency and execution speed compared to the system processing all data in the secure area. We need to work on designing more query support and other important modules in the future.

REFERENCES

[1] C. Priebe, K. Vaswani, and M. Costa, "Enclavedb: A secure database using SGX," in *IEEE S&P 2018*, pp. 264–278, IEEE Computer Society, 2018.
[2] D. Vinayagamurthy, A. Gribov, and S. Gorbunov, "Stealthdb: a scalable encrypted database with full SQL query support," *Proc. Priv. Enhancing Technol.*, vol. 2019, no. 3, pp. 370–388, 2019.
[3] Y. Wang, Y. Shen, C. Su, J. Ma, L. Liu, and X. Dong, "Cryptsqlite: Sqlite with high data security," *IEEE Transactions on Computers*, vol. 69, no. 5, pp. 666–678, 2019.
[4] M. Taassori, A. Shafiee, and R. Balasubramonian, "Vault: Reducing paging overheads in sgx with efficient integrity verification structures," in *ASPLOS'18*, pp. 665–678, 2018.
[5] O. Weisse, V. Bertacco, and T. Austin, "Regaining lost cycles with hotcalls: A fast interface for sgx secure enclaves," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 81–93, 2017.
[6] Y. Xu, W. Cui, and M. Peinado, "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," in *IEEE S&P 2015*, pp. 640–656, 2015.
[7] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller, "Cache attacks on intel sgx," in *Proceedings of the 10th European Workshop on Systems Security*, pp. 1–6, 2017.
[8] "Intel Software Guard Extensions (Intel SGX) SDK for Linux OS." https://download.01.org/intel-sgx/sgx-linux/2.13/docs/Intel_SGX_Developer_Reference_Linux_2.13_Open_Source.pdf.
[9] H. Tian *et al.*, "Switchless calls made practical in intel sgx," in *SysTEX'18*, pp. 22–27, 2018.
[10] M. Orenbach, P. Lifshits, M. Minkin, and M. Silberstein, "Eleos: Exitless os services for sgx enclaves," in *EuroSys'17*, pp. 238–253, 2017.
[11] S. Shinde, Z. L. Chua, V. Narayanan, and P. Saxena, "Preventing page faults from telling your secrets," ASIA CCS '16, (New York, NY, USA), p. 317–328, Association for Computing Machinery, 2016.
[12] F. Brasser *et al.*, "Software grand exposure: Sgx cache attacks are practical.," in *WOOT*, pp. 11–11, 2017.
[13] M.-W. Shih, S. Lee, T. Kim, and M. Peinado, "T-sgx: Eradicating controlled-channel attacks against enclave programs.," in *NDSS*, 2017.
[14] A. Rane, C. Lin, and M. Tiwari, "Raccoon: Closing digital {Side-Channels} through obfuscated execution," in *24th USENIX Security Symposium (USENIX Security 15)*, pp. 431–446, 2015.

 ISBN: 978-1-68558-056-8