FPGA-accelerated Searchable Encrypted Database Management Systems for Cloud Services

Mitsuhiro Okada, Takayuki Suzuki, Naoya Nishio, Hasitha Muthumala Waidyasooriya, and Masanori Hariyama

Abstract—The use of database management systems (DBMSs) as a cloud service is rapidly expanding. Cloud DBMSs offer many advantages, such as easier management, lower costs, and greater scalability. However, there are still security concerns regarding attacks from adversaries. DBMSs that use searchable encryption have been investigated with regard to ensuring their security. Because searchable encryption allows query execution over encrypted data in the cloud, sensitive data can be securely stored there in the cloud. On the other hand, encrypted query processing is slower than query processing on plaintext data. In this paper, we use a field-programmable gate array (FPGA) to accelerate query processing in a searchable encrypted DBMS. We also propose a new cache function to shorten the access time to database tables in a DBMS. According to an evaluation using basic queries, the proposed system has achieved up to 110.7 times speed-up compared with central processing unit (CPU) processing of a single core. In addition, the proposed system can process queries faster than the plaintext processing on a CPU when processing large amounts of data.

Index Terms—Database management system, FPGA, Searchable encryption, OpenCL.

1 Introduction

The use of database management systems (DBMSs) as a cloud service is rapidly expanding. The benefits of cloud DBMSs include easier management, lower costs, and greater scalability. However, while cloud DBMSs offers many advantages to users, they also introduce security concerns. For example, there was an incident in which a cloud administrator leaked sensitive personal information [1] and another incident in which a hacker leaked personal information by obtaining administrative privileges [2], [3].

One means of protecting sensitive data from adversaries is to encrypt data on the client side using encryption functions provided by a DBMS [4], [5]. In this approach, however, a DBMS can execute only equality queries because the data stored on the DBMS server is encrypted with a deterministic algorithm on the client side. If a user requests other query types, all encrypted data must be decrypted on the client side after being downloaded from the cloud. This extra process undermines the convenience and efficiency of cloud services.

Another approach is to arrange a secure trusted piece of hardware on the cloud that the cloud administrator cannot access. Because encrypted data are decrypted using a secret key stored in the trusted hardware, queries can be executed securely [6], [7], [8], [9]. However, this requires the assumption that the trusted hardware is perfectly secure. If the secret key is stolen by adversaries as a result of hardware bugs, backdoors, or side-channel attacks, all the encrypted data in the database are decrypted. Indeed, one recent study [10] succeeded in extracting the secret key from trusted hardware using a side-channel attack, thereby showing that security is not guaranteed on this approach.

The use of DBMSs that employ searchable encryption to overcome these problems has been proposed [11], [12], [13]. Such a DBMS (which we call a searchable encrypted DBMS) enables query execution over encrypted data in the cloud while protecting sensitive data from adversaries. Nevertheless, a searchable encrypted DBMS has two drawbacks with respect to computational performance. First, query execution on encrypted data requires unique computations that slow down the processing speed compared to plaintext data processing. Second, access time to database tables increases as the amount of data increases because of the encryption process.

Recently, cloud vendors such as Amazon [14], IBM [15], and Microsoft [16] have introduced field-programmable gate arrays (FPGAs) that aim to accelerate user specific processing in the cloud. In this paper, we propose an FPGA accelerator and system architecture for accelerating a searchable encrypted DBMS. We also propose a new cache function called Crypto Cache to reduce the access time to database tables in a DBMS.

• *M. Okada and N. Nishio are with Center for Technology Innovation - Digital Technology, Research & Development Group, Hitachi, Japan.*
E-mail: {mitsuhiro.okada.uf, naoya.nishio.jv}@hitachi.com
• *T. Shzuki is with Financial Institutions Business Unit, Hitachi, Japan.*
E-mail: takayuki.suzuki.gu@hitachi.com
• *M. Okada, H. Waidyasooriya and M. Hariyama are with Tohoku University, Japan.*
E-mail: mitsuhiro.okada.r5@dc.tohoku.ac.jp, {hasitha, hariyama}@ecei.tohoku.ac.jp

2 Related Work

There are some studies [50], [51], [52] about accelerators for searchable encryption processing. However, these studies were intended only to speed-up the searchable encryption processing and not evaluated with DBMSs.

Table 1 summarizes the comparison of this work with previous work in relation to DBMSs. Although accelerators for DBMSs and searchable encrypted DBMSs are both reported in previous work, there is no study combining the two ideas. Our system is the first to use an FPGA to accelerate encrypted query processing. This section introduces previous work on accelerators for DBMSs and searchable encrypted DBMSs.

TABLE 1
Comparisons of previous studies and our work

	[17], [18], [19], [20], [21], [22], [41], [42], [53], [54], [55], [56]	[11], [12], [13], [23], [24], [25]	This work
Accelerator	Supported	Not supported	Supported
Searchable Encryption	Not supported	Supported	Supported

Accelerators for DBMSs. Netezza [17] combines an FPGA with a hard disk drive to accelerate sequential scan processing. Dennl et al [18] design an FPGA accelerator to perform *WHERE* clause processing in parallel. Some studies [19], [20], [21], [22] accelerate *join* operations using an FPGA or a graphical processing unit (GPU). OmniDB [41] and Postgresql [42] support GPU acceleration for subsets of Structured Query Language (SQL). Wang et al [53] accelerated some operations by generating feasible execution plans with multiple FPGA images on OpenCL-based FPGAs. Furthermore, FPGAs have been used for studies of near storage processing to accelerate DBMSs [54], [55], [56].

Because the purpose of these studies is to accelerate subsets of SQL in a DBMS, they do not address the acceleration of searchable encrypted DBMSs.

Searchable Encrypted DBMSs. The pioneering work on searchable encrypted DBMSs involved CryptDB [11]. CryptDB uses several encryption schemes to create a DBMS, supporting *equality*, *order*, *word search*, *addition*, and *join* operations. MONOMI [23], which builds on CryptDB, improves the performance on the Transaction Processing Council Benchmark H (TPC-H) [49] by optimizing the partitioning of query execution between the client and the server. Other methods of improving the performance of CryptDB include the acceleration of the Advanced Encryption Standard (AES) processing that is part of encrypted query processing by using AES hardware in a CPU [24] and the acceleration of *addition* operations by applying the Chinese remainder theorem [25].

Seabed [12] and Arx [13] enhance security or improve the processing speed as compared with CryptDB. Seabed uses order-preserving encryption [33] for *order* operations, a more secure encryption scheme than the one used in CryptDB, and also uses an additively symmetric homomorphic encryption scheme for accelerating *addition* operations. Arx uses Arx-RANGE based on tree traversal for *order* operations and probabilistic encryption for *equality* operations to strengthen security.

Although these studies attempt to improve query processing speed, their approaches are limited to optimizing algorithms and using AES hardware in a CPU. These studies do not consider the use of specialized hardware for acceleration.

3 Proposed Searchable Encrypted DBMS

3.1 System architecture

Fig. 1 shows the system architecture of the proposed FPGA accelerated searchable encrypted DBMS. Following the architecture of Arx, our architecture deploys an application proxy at the application server and a server proxy at the DBMS server. Because encrypted query processing is performed in these two proxies, the application and the DBMS does not require modifications. The new components of our architecture are the Query Accelerator on the FPGA and the Crypto Cache on the server proxy. The FPGA board is an external device connected to the DBMS server via Peripheral Component Interconnect Express (PCIe), with the result that said Query Accelerator cannot access the memory on the DBMS server directly. This creates overhead in that data must be transferred from the memory on the DBMS server to the memory on the FPGA board. However, when the Query Accelerator is required to repeatedly perform the same computation over large amount of data, it can process faster than a CPU because FPGAs use parallelism and pipeline processing.

Following are the explanations of each module in Fig. 1.

Query Handler. The Query Handler has two functions. The first is to perform encryption-related functions (e.g., managing the secret key, encrypting and decrypting data, or generating encrypted queries). Fig. 2 shows an example of an encrypted table. Not only values, but also table names and column names are encrypted. Table names and column names are encrypted to ensure that the frequency distribution of encrypted data will not be guessed from those names.

The second function is to convert a normal SQL query issued by the application into a query plan that will be executed in the server proxy based on the metadata of encrypted tables. These metadata are generated at the same time as the creation of encrypted tables. Fig. 3 shows an example of a query plan. A query plan is written in JavaScript Object Notation (JSON) format and sent to the cloud with not only the search condition but also the table and column names being encrypted.

Crypto Cache. The Crypto Cache also has two functions. The first is to read encrypted tables from the DBMS and store them in the memory managed by the server proxy. This function enables the Query Executor to do without access to the DBMS when it creates a result table to be returned to the application server.

The second function is for advance setting of data for predicate processing. The Crypto Cache removes unnecessary data from the encrypted tables and transfers the organized data to the memory on the FPGA board.

These two functions of the Crypto Cache are executed on startup. In cases in which queries that update encrypted tables in the DBMS (e.g., *INSERT*, *DELETE*, and *UPDATE*) are issued, the corresponding data in both the DBMS and the Crypto Cache are updated accordingly to ensure the consistency of data between the DBMS and the Crypto

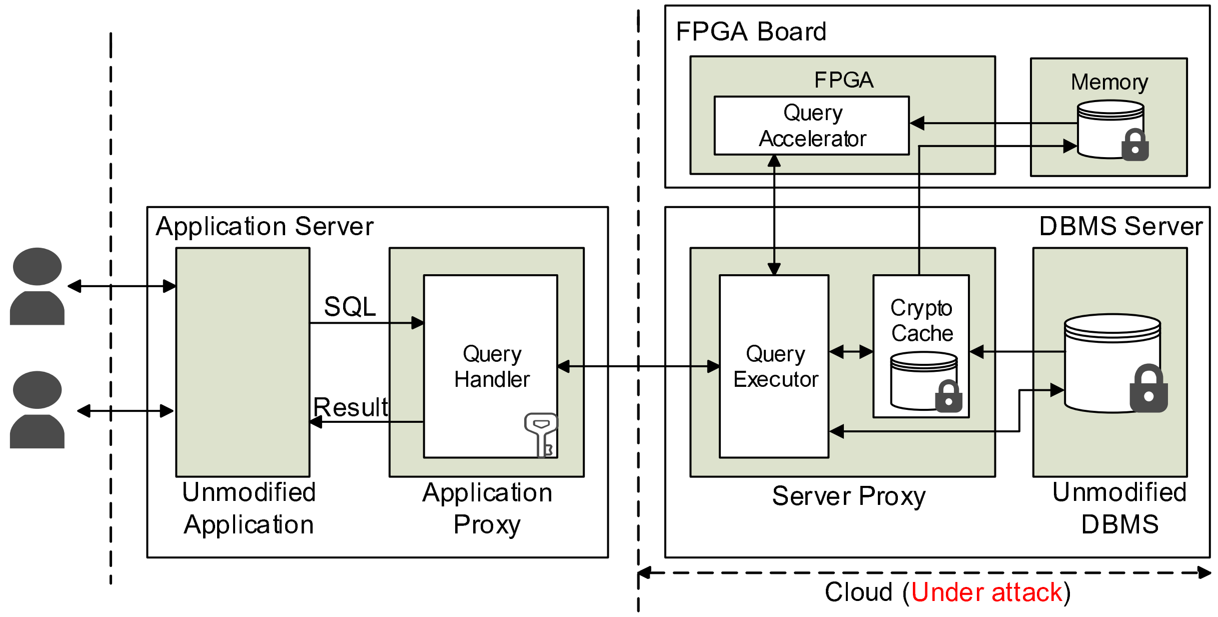


Fig. 1. Overview of the system architecture

Enc(Name)	Enc(age)
Enc(Alice)	Enc(20)
Enc(Bob)	Enc(30)

Fig. 2. Example of an encrypted table

```
[
  {
    "Plan": {
      "Table name"      : "Enc(Table A)",
      "Operation"       : "Equality check",
      "Predicate"       : ["Enc(Age)" = "Enc(30)"],
      "Output Columns"  : ["Enc(Name)", "Enc(Age)"]
    }
  }
]
```

Fig. 3. Example of a query plan

Cache. Consequently, the Query Executor has no need to read encrypted tables from the DBMS every time it receives a query plan from the application server. While normally a DBMS itself has a function to cache tables for the same queries, we implemented the Crypto Cache to bypass the process of reading encrypted data from the DBMS and transferring the data after organizing the data to the memory on the FPGA board.

Query Executor. The Query Executor receives a query plan and executes it. Because encrypted data are already on the FPGA board owing to the Crypto Cache, the Query Executor is required only to send an encrypted search condition to the memory on the FPGA board. Once the encrypted data and the encrypted search condition are both in the memory, the Query Executor orders the Query Accelerator to perform predicate processing. The processing results are first stored in the memory on the FPGA board and then transferred to the memory in the server proxy. Based on the processing results, the Query Executor obtains the encrypted data in output columns requested by the query plan from the Crypto Cache and returns the data to the application proxy.

Query Accelerator. The Query Accelerator accelerates predicate processing. This component is described in detail in Section 3.3.

The architecture explained above enables query execution without decrypting the encrypted data on the DBMS server. In addition, the architecture is designed in such a way that the proposed FPGA accelerator for cloud DBMSs can be implemented without modifying the DBMS. In other words, we designed the architecture to be independent of the choice of DBMS. Thus, our proposed system can be implemented easily on various DBMSs just by changing the server proxy according to the application programming interface (API) of the DBMS.

3.2 Functionality

We implemented four basic DBMS query operations, *equality*, *order*, *word search*, and *join*, because these basic query operations are most often used for highly sensitive data, such as lists of personal information. In the TPC-H benchmark, under the experimental conditions in Section 5, 90% of queries related to the encrypted table can be handled by these four basic operations. We will discuss support for other operations such as *addition* and *ORDER BY* in Section 6.

Equality. Various studies proposed searchable encryption schemes that support *equality* operations [26], [27], [28], [29], [30]. We use Symmetric Searchable Encryption (SSE), proposed by Yoshino et al [29]. We chose SSE because it uses different pseudorandom functions for encrypting data and encrypting search conditions, making their scheme the most secure compared to the others.

Order. There are several studies on searchable encryption schemes that are compatible with *order* operations [31], [32], [33], [34], [35]. We use Order-Revealing Encryption (ORE), proposed by Lewi et al [34], which provides both high-level security and practical processing speed. In terms of security, the method using Arx-RANGE in Arx is more secure than ORE, but the increase in data size as a result of encryption is far greater with garbled circuits, rendering the method less desirable for large-scale databases. We used the source code of ORE available on GitHub [36] with the parameters set as Bit Length = 32; Block Size = 8. Note that the data encrypted with ORE are used only for *order* operations and do not get decrypted. To reduce the storage capacity, only the columns required for *order* operations are stored when a user creates a new encrypted table.

Word Search. We support three subfunctions of *word search*: *prefix search*, *substring search*, and *suffix search*. As for *word search* operations, approaches that use k-grams and suffix trees to perform searches efficiently have been proposed [37], [38], [39], [40]. However, we chose a method that separates plaintexts into single characters and uses the SSE to encrypt each character, because we can implement this method in parallel on FPGAs by using pipelines. To perform the *word search*, the FPGA accelerator requires the boundary information of each field. Because this information is created when Crypto Cache reads encrypted tables from the DBMS, the information is not required to be stored in the DBMS. Note that the encrypted data of each character are used only for *word search* operations and do not get decrypted. To reduce the storage capacity, only the columns required for *word search* operations are stored when a user creates a new encrypted table.

Join. Our system supports *equality-join* operations using SSE. However, because we use probabilistic encryption that makes use of a pseudorandom function for data encryption, *equality* operations cannot be performed between encrypted data items. To address this problem, for columns in which *join* operations are executed, an encrypted search condition that corresponds to each encrypted datum is stored in the database. Even though this encrypted search condition is stored in the database, it is encrypted using a pseudorandom function as well and contains no information that could potentially be used to decrypt the encrypted data.

3.3 FPGA implementation

3.3.1 Overview

Fig. 4 shows an overview of the FPGA accelerator using Intel FPGA for OpenCL [44]. The accelerator consists of a BSP and programmable logic for user-created modules. The BSP contains fixed resources that communicate with external devices and control the modules. The programmable logic includes five modules connected using four first-in and first-out (FIFO). The circuit design in programmable logic is written in the C programming language. The circuit is generated from the C source code using the HLS Compiler [44]. Consequently, even software engineers who have never designed a circuit can design logic modules. Moreover, engineers can easily control the FPGA accelerator using the OpenCL API of (1)–(3) in Fig. 4.

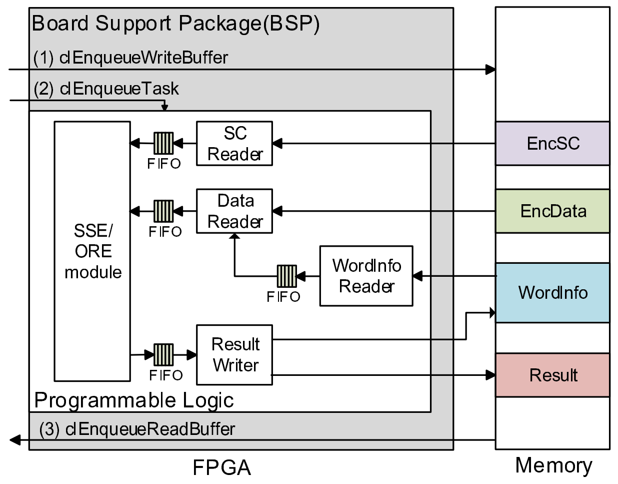


Fig. 4. Overview of the FPGA accelerator

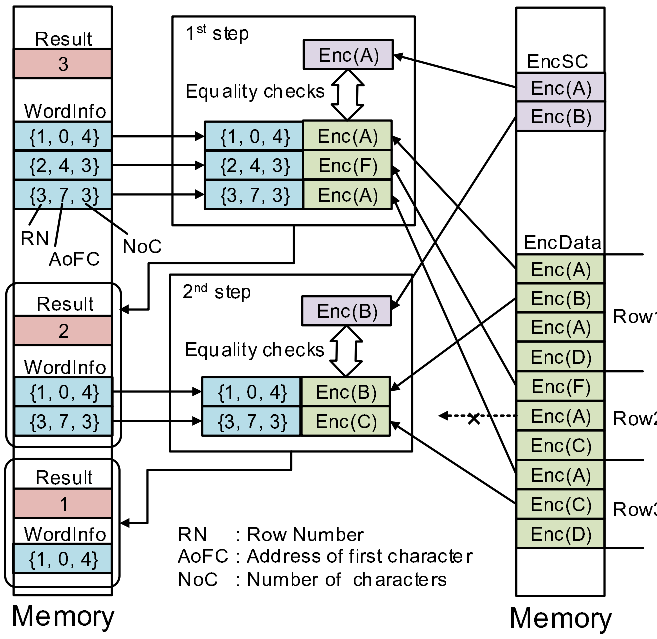


Fig. 5. Example of a prefix search: The search for the prefix "AB" is executed on three rows of data, "ABAD," "FAC," and "ACD."

3.3.2 Implementation detail in Programmable Logic

Reader and Writer modules. The Reader and Writer modules work differently for *word search* and other operations. Each case is described below.

When performing *equality*, *order*, or *join* operations, the Data Reader reads encrypted data (EncData) in all rows in the order of address. The SC Reader reads encrypted search conditions (EncSC). The SSE or ORE module is executed if EncData and EncSC are present in FIFO. The Result Writer stores the information of match(1)/not match(0) for every row in the Result region. The WordInfo Reader is not used.

When performing *word search* operations, an algorithm that narrows down the calculation candidates is used to reduce the amount of computation. Fig. 5 shows an example of how the Reader and Writer modules work in a prefix search. In this example, a query that searches for prefix

"AB" is executed on three rows of data: "ABAD," "FAC," and "ACD." The encrypted data of each character is stored in the EncData region. The encrypted search condition of each character is stored in the EncSC region. As information for accessing specific EncData, the field information (row number (RN), address of the first character (AoFC), and number of characters (NoC)) of each row are stored in the WordInfo region.

At the first step, WordInfo Reader reads the field information from the WordInfo region. Based on the AoFC in the field information, Data Reader reads the first character of EncData in each row, and SC Reader reads the first character of EncSC. After the SSE module performs equality checks using both EncSC and EncData, Result Writer writes only the field information of the matched rows in the WordInfo region and writes the total number of matched rows in the Result region.

At the second step, WordInfo Reader reads field information written by the previous step, Data Reader reads the second character of EncData based on the AoFC in the field information, and SC Reader reads the second character of the EncSC. Result Writer writes only the field information of the matched rows in the WordInfo region and writes the total number of matched rows in the Result region.

These steps are repeated as many times as the NoC in the search conditions. At the completion of processing, the server proxy extracts the RN from the field information written in the WordInfo region.

Although the example given above is of a prefix search, substring and suffix searches can be executed by changing the behavior in the first step of WordInfo Reader. To be more precise, in suffix searches, the AoFC in the field information is rewritten by calculating it from the NoC and the number of search conditions. In substring searches, the field information is replicated after incrementing the AoFC by one to perform equality checks while shifting one character.

SSE module. SSE performs equality checks with EncData and EncSC as inputs in the manner described in Fig. 6, where kw is plaintext data, qr is a search condition, r and r' are different pseudorandom values, sk_e is the secret key, and sk_s is the key shared with the application server. We implement AES in a cipher block chaining mode as encryption function $E()$, SHA256 as hash function $H()$, and the MixColumn operation in Rijndael [46] as the homomorphic function $F()$. If kw and qr are equal, the two pseudorandom values r' in EncSC cancel each other. Consequently, the computation result is equal to $H(F(r))$ in EncData. SSE is designed to perform pipeline processing, in which decryption and $F()$ are executed in parallel.

ORE module. Now, we explain the processing procedure of ORE (Fig. 7). ORE performs order checks with EncData and EncSC as inputs. $ctxt1\_l$, $ctxt2\_l$, and $ctxt2\_r$ each consists of four blocks. $Nonce$, the key shared with the application server, is stored in EncSC. Blocks in the same positions in $ctxt1\_l$ and $ctxt2\_l$ are tested for equality, and if the two blocks differ, ORE performs the calculations shown in Eqs. (1) and (2) to determine whether EncData are larger or smaller than EncSC. If there are no differing blocks between $ctxt1\_l$ and $ctxt2\_l$, they are judged to be equal.

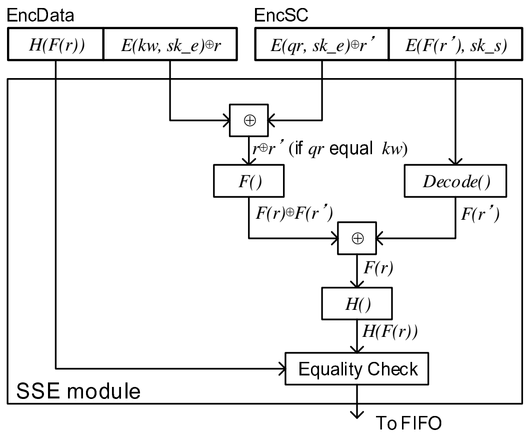


Fig. 6. Design of the SSE predicate processing module

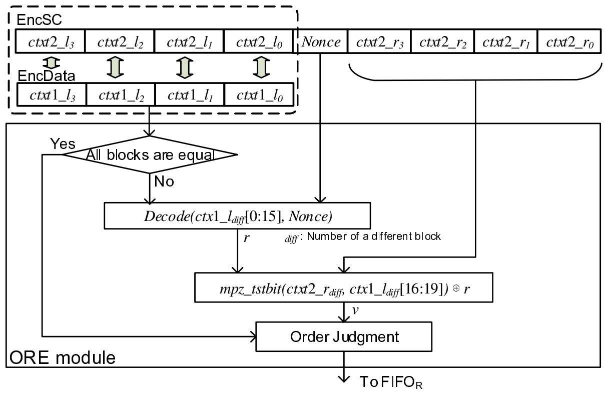


Fig. 7. Design of the ORE predicate processing module

$$r = Decode(ctx1\_l_{diff}[0:15], Nonce) \tag{1}$$

$$v = mpz\_tstbit(ctxt2\_r_{diff}, ctx1\_l_{diff}[16:19]) \oplus r \tag{2}$$

In these equations, $diff$ indicates the number of different blocks. We implemented AES in a counter mode for Decode and the mpz_tstbit [48] function in the GNU Multiple Precision Arithmetic Library for mpz_tstbit. ORE is also designed to perform pipeline processing, in which equality checks between pairs of blocks are executed in parallel.

3.3.3 Implementation result

We implemented the above-described functions using the FPGA board and FPGA compiler described in Table 2. Table 3 shows the FPGA resource utilization of each logic module. The numbers in parentheses indicate the respective percentages of resources used by each of the modules.

Each module operates at 213 MHz. The power consumption of the FPGA is 26.4 W, which was calculated using the power estimation tool provided by Intel [57]. Although the clock frequency of the FPGA is lower than that of the CPU, it can process at a higher speed on lower power because the FPGA has the capabilities of pipelining and parallel processing.

TABLE 2
Machine specifications

Items	Specification
CPU	Intel Xeon Silver 4110@2.1GHz
Server memory	DDR4 2400MHz 192GB
FPGA board	Intel Arria10 evaluation board (DK-DEV-10AX115S-A) with DDR3 1066MHz 2GB
OS	Ubuntu 16.4 LTS
DBMS	MySQL 5.7.22
FPGA compiler	Intel SDK for OpenCL version 17.1

TABLE 3
Resource utilization

Module	ALUTs	FFs	RAMs	DSPs
BSP	80771	166418	367	0
Data Reader	14832	20849	48	2
SC Reader	7175	10147	26	2
WordInfo Reader	7608	10051	32	0
SSE	68146	52468	218	2
ORE	12544	18289	223	0
Result Writer	25839	29000	56	6
Total	216915 (28%)	307222 (20%)	970 (41%)	12 (1%)

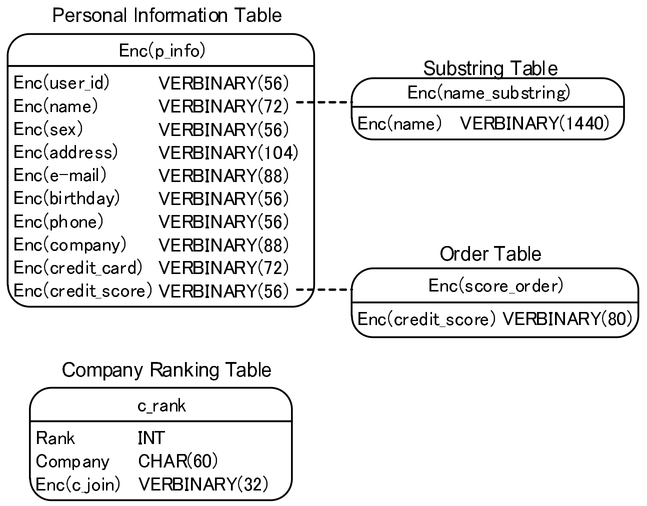

Personal Information Table
Enc(p_info)
Enc(user_id) VERBINARY(56)
Enc(name) VERBINARY(72)
Enc(sex) VERBINARY(56)
Enc(address) VERBINARY(104)
Enc(e-mail) VERBINARY(88)
Enc(birthday) VERBINARY(56)
Enc(phone) VERBINARY(56)
Enc(company) VERBINARY(88)
Enc(credit_card) VERBINARY(72)
Enc(credit_score) VERBINARY(56)
Substring Table
Enc(name_substring)
Enc(name) VERBINARY(1440)
Order Table
Enc(score_order)
Enc(credit_score) VERBINARY(80)
Company Ranking Table
c_rank
Rank INT
Company CHAR(60)
Enc(c_join) VERBINARY(32)

Fig. 8. Structure of the test table

TABLE 4
Comparison of the total amount of data between plaintext and ciphertext with 1 M rows.

Table name	Plaintext	Ciphertext
Personal information table	230 MB	692 MB
Word search table	-	1392 MB
Order table	-	84 MB
Company Ranking table	0.64 KB	0.96 KB
Total	230 MB	2068 MB

4 Evaluation

4.1 Evaluation setup

Machine specifications. We assessed the effectiveness of our FPGA accelerator by implementing an application server and a DBMS server in a single machine. Table 2 shows the hardware and software we used.

Test table. We chose a list of personal information as our test data as shown in Fig. 8. We used Faker [45] to generate a list of personal information. The ten columns in the personal information table are user ID, name, sex, address, e-mail, birthday, phone, company, credit card, and credit scores, which are all encrypted. The word search table used for *word search* and the order table of credit scores used for *order* operations are stored as tables separated from the personal information table. The information on the link between the personal information table and the other tables is managed by the Query Handler as metadata. We also created a company ranking table, which is used for evaluating *join* operations. This table stores the information of company names with top ten revenues and their rankings. The company names and their rankings are stored as plaintext. Encrypted search conditions that correspond to each company are also stored in the same table to be used for *join* operations.

Table 4 shows a comparison of the total amount of data between plaintext and ciphertext with 1 M rows. The total data volume of the ciphertext tables increases approximately nine-fold after encryption.

Test queries. The test queries are the five basic queries listed in Table 5. The number of *equality* operations in *prefix search* and *suffix search* is the same as in our FPGA implementation. Consequently, we evaluate the performance only of *prefix search*. We discuss support for queries with multiple search conditions in Section 6.

Evaluation method. We set the performance of CPU processing with the database cache (CPU w/DBC) as the baseline. To check the acceleration effect of both the Query Accelerator and the Crypto Cache, we evaluate FPGA processing with the database cache (FPGA w/DBC), CPU processing with the Crypto Cache (CPU w/CC), and FPGA processing with the Crypto Cache (FPGA w/CC). Note that CPU w/CC does not use the function to transfer encrypted data to the memory on the FPGA board. We also evaluate plaintext data processing in the CPU (Plain). We measure the processing time of end-to-end latency, which is the duration from the point at which the application proxy receives an SQL query to the point at which the application proxy prepares plaintext data for the user.

In CPU processing, the CPU in the server proxy performs predicate processing with SSE or ORE as described in Section 3. CPU processing in all setups is performed using a single core CPU. The end-to-end latency of plaintext data processing is measured without using the database cache.

The encrypted tables are stored in the memory in the server proxy using the in-memory storage engine of MySQL. We evaluate performances for tables with 100, 1 K, 10 K, 100 K, 500 K, 1 M and 10 M rows because the end-to-end latency changes depending on the number of rows in a table. Note that owing to the insufficient memory capacity on the FPGA board, *prefix search* and *substring search* with 10 M rows using FPGA are not executed.

TABLE 5
List of test queries

Operation type	Query
Equality	*SELECT * FROM* p_info *WHERE* use_id = 10;
Order	*SELECT * FROM* p_info *WHERE* credit_score > 845;
Prefix	*SELECT * FROM* p_info *WHERE* name *LIKE* "Sherry%";
Substring	*SELECT * FROM* p_info *WHERE* name *LIKE* "%Sherry%";
Join	*SELECT* p_info.user_id, p_info.name, c_rank.rank *FROM* p_info *INNER JOIN SUB ON* p_info.company = c_rank.company;

TABLE 6
End-to-end latency for 1M rows

	Plaintext	CPU w/DBC	FPGA w/CC	Speed-up
Equality	46.3 [ms]	942.9 [ms]	8.5 [ms]	x110.7
Order	58.2 [ms]	646.6 [ms]	24.9 [ms]	x26.0
Prefix	75.5 [ms]	1362.7 [ms]	16.3 [ms]	x83.6
Substring	91.7 [ms]	4949.8 [ms]	76.7 [ms]	x64.5
Join	968.6 [ms]	5503.4 [ms]	83.7 [ms]	x65.7

4.2 Performance evaluation

Table 6 shows the end-to-end latency of tables with 1 M rows, where the ratio of the speed-up become stable for all operations. FPGA w/CC is 26.0 to 110.7 times faster than CPU w/DBC and also achieved faster than plaintext processing in the CPU. This indicates that storing sensitive data safely in a cloud DBMS while minimizing performance degradation is possible. We investigate in more detail from the following two perspectives.

End-to-end latency against the number of rows. For Figs. 9(a)–13(a), we present the graph of end-to-end latency plotted against the number of rows. If we compare w/DBC and w/CC for each of CPU processing and FPGA processing, w/CC is always faster than w/DBC, which confirms the effectiveness of the Crypto Cache. On the other hand, regarding the Query Accelerator, the effectiveness of acceleration differs depending on the number of rows. For example, comparing the effect of the speed-ups of CPU w/CC and FPGA w/CC, the effect of the speed-up is large and small at 1 M and 100 rows, respectively. The reason for this difference is the control overhead detailed in the breakdown of end-to-end latency. The results of the performance evaluation demonstrate that our FPGA accelerator is highly effective when the number of rows is sufficiently large that the control overhead becomes negligible.

In addition, we can observe that for greater than 100 K rows, end-to-end latency increases nearly in proportion to the number of rows. This suggests that the effectiveness of the FPGA accelerator persists even if the number of rows increases further. We believe that if we can prepare sufficient memory for storing data in our system, query execution on even a large-scale encrypted database with tables of 1 or 10 M rows or more can be performed while obtaining the same speed-up ratio as in Table 6. We will discuss ways to overcome the problem of memory capacity in Section 6.

Breakdown of end-to-end latency. The breakdown of end-to-end latency for operations on 100 rows and 1 M rows is shown in Figs. 9(b)–13(b) and Figs. 9(c)–13(c), respectively. The end-to-end latency of each operation is divided into six different processes: encrypting a query and creating a query plan (Encrypt), reading encrypted data from the DBMS (Read data), transferring data from the server proxy to FPGA (FPGA Trans.), performing predicate processing with the CPU (CPU Proc.) or with the FPGA (FPGA Exe.), preparing an encrypted table with query results for the user (Prepare table), and decrypting the encrypted table (Decrypt). We can see that the proportions occupied by the six processes differ, according to the operation type and the number of rows.

In Figs. 9(b)–13(b) and Figs. 9(c)–13(c), when the Crypto Cache is not used, Read data (gray bar), FPGA Trans. (pink bar) and Prepare table (green bar) are required. The green bar reads the personal information table necessary for returning query results from the DBMS, differing from the *equality* and *join* queries.

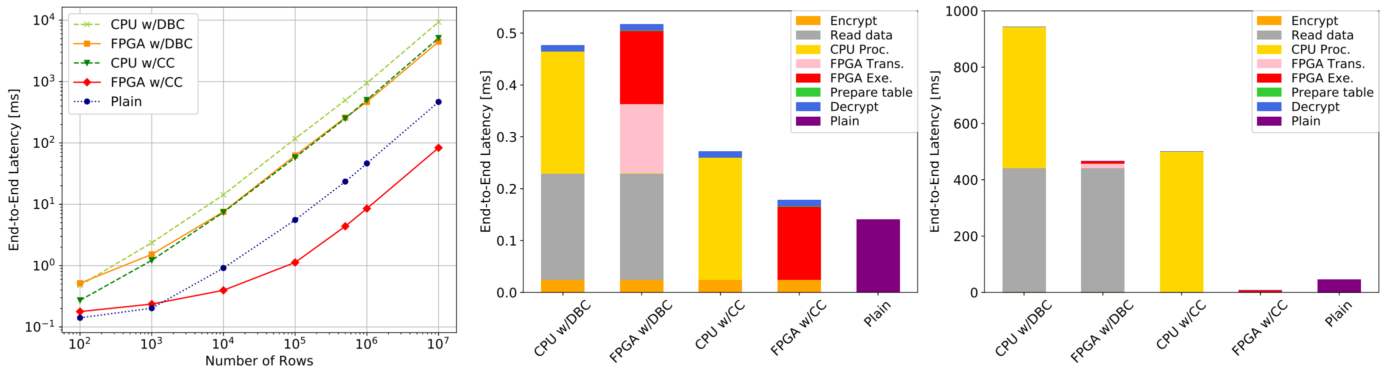
Focusing on CPU w/DBC in Figs. 9(c)–13(c), the proportion of predicate processing time (yellow bar) to the read data time (gray bar) and the prepare table time (green bar) is large for *substring* and *join* queries because the same EncData are subjected to *equality* operation processing under different search conditions.

As shown in Fig. 10(c), a large portion of query processing time with FPGA w/CC is occupied by decrypting. This is the reason that a relatively small speed-up is achieved for *order* queries compared to the other queries, as shown in Table 6. Actually, the decrypting time of single encrypted data are extremely small (approximately 0.36 ns). However, when the amount of encrypted data to be decrypted is considerably large, the decrypting time can be a bottleneck. To address this problem, we can use an FPGA accelerator in the application proxy for decryption.

The breakdown of the 100 rows (Figs. 9(b)–13(b)) shows that the FPGA processing (FPGA Trans. and FPGA Exe.) is slower or slightly faster than the CPU Proc.; no speed-up is as effective as that of 1 M rows because both FPGA Trans. and FPGA Exe. have fixed control overheads in addition to the data transfer time and FPGA execution time proportional to the number of rows. For 100 rows, the overall processing time is as short as approximately 0.5-2 ms; therefore, we consider that this control overhead has significantly affected the FPGA processing time because the time of processing proportional to the number of rows is very small.

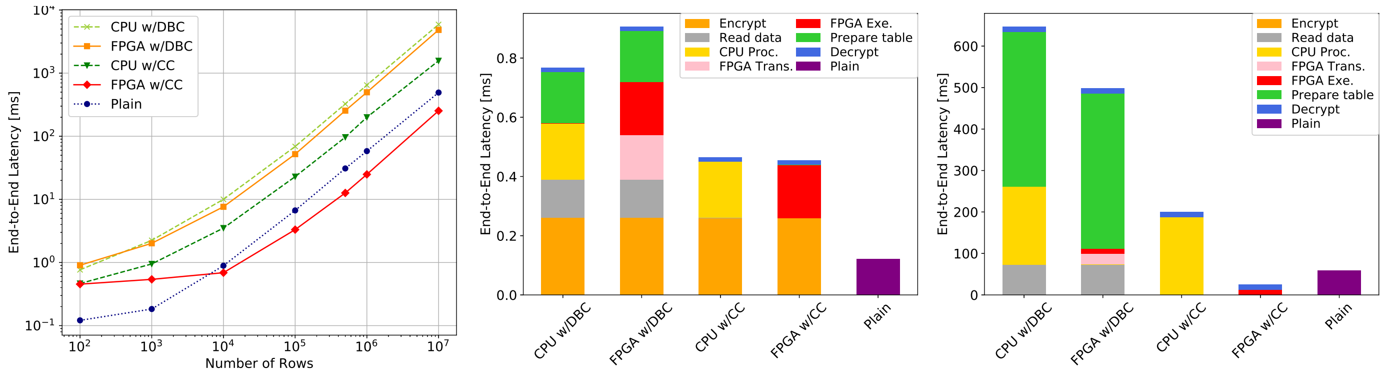
4.3 Performance comparison of multi-thread CPU and FPGA

We compared the performance of multi-threaded CPU w/CC and FPGA w/CC with 1 M rows. The evaluation



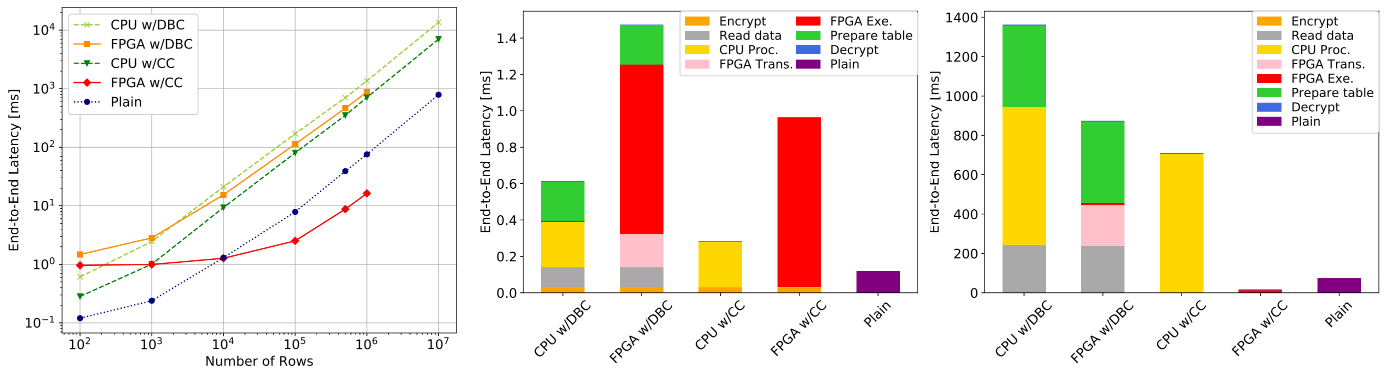
(a) End-to-end latency (b) Breakdown of latency for 100 rows (c) Breakdown of latency for 1 M rows

Fig. 9. Result of *equality* operations



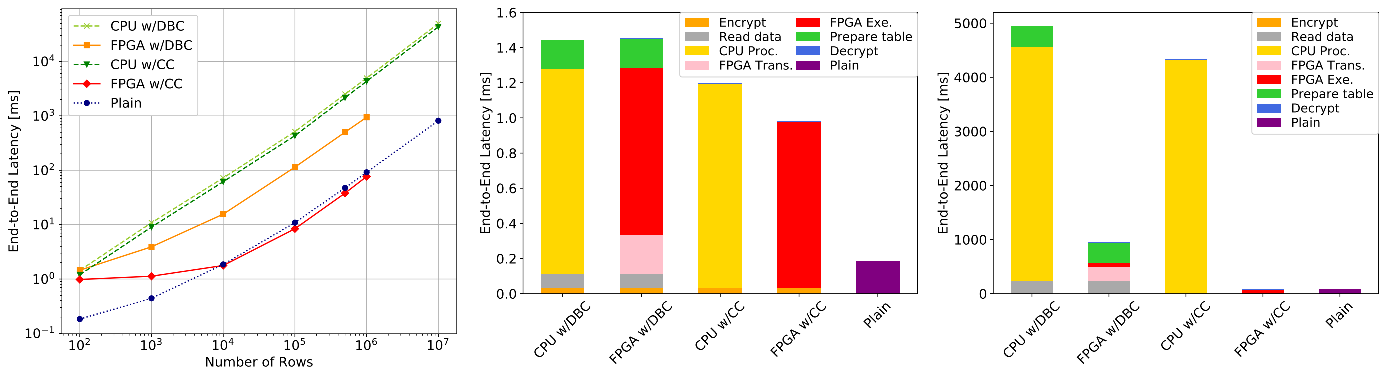
(a) End-to-end latency (b) Breakdown of latency for 100 rows (c) Breakdown of latency for 1 M rows

Fig. 10. Result of *order* operations



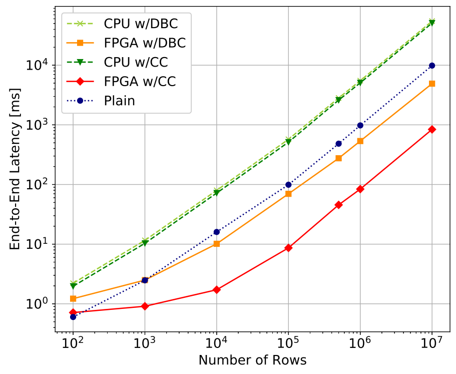
(a) End-to-end latency (b) Breakdown of latency for 100 rows (c) Breakdown of latency for 1 M rows

Fig. 11. Results of *prefix search*

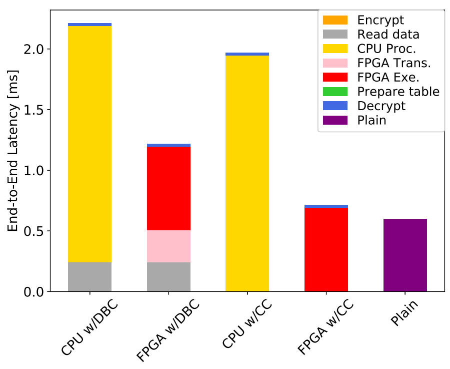


(a) End-to-end latency (b) Breakdown of latency for 100 rows (c) Breakdown of latency for 1 M rows

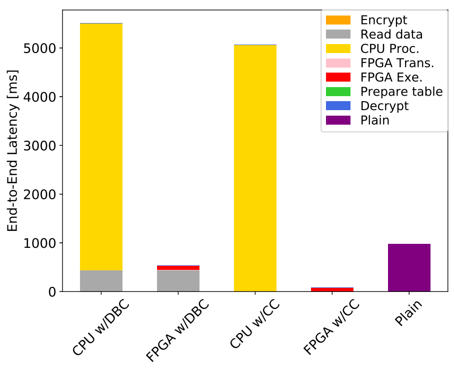
Fig. 12. Results of *substring search*



(a) End-to-end latency



(b) Breakdown of latency for 100 rows



(c) Breakdown of latency for 1 M rows

Fig. 13. Results of *join* operations

of the multi-thread performance was performed for up to 16 threads by use of the thread function in C++11. In Figs. 9(c)-13(c), only the CPU Proc. (yellow bar) is multi-threaded.

Table 7 shows the results of the multi-threading with 1 M rows. We can observe that the processing speed increases as the number of threads increase but saturates at 16 threads because of the 8-core CPU. With this result, we confirmed that FPGA w/CC is faster than CPU w/CC when using 16 threads.

In addition, the power consumptions of the CPU and the FPGA were compared. The power consumption of the Xeon is 85 W from the Thermal Design Power value in the specification sheet. However, the power consumption of the FPGA is 26.4 W as stated in Section 3.3.3. These findings imply that the FPGA has lower power consumption than the CPU. If the number of CPUs is increased, the processing speed increases and the power consumption accordingly increases. Therefore, we think that the FPGA has good computational resources in terms of power efficiency.

4.4 Comparison with other searchable encrypted DBMSs

4.4.1 Performance evaluation

We compare the performance of our proposed system with other systems for processing 1 M rows of data. The systems compared here include CryptDB, Arx, and Seabed, all of which are searchable encrypted DBMSs. Because *word search* operations are not supported in these three systems, we compare the performance of thier *equality*, *order*, and *join* operations in Table 8.

Using the source code available on GitHub [47], we implemented CryptDB on the same machine as ours and measured the end-to-end latency using the same test table. When CryptDB executes a query, outer layers of probabilistic encryption are removed, leaving only the layer of deterministic encryption to protect data. As a result, the security level decreases, but the query processing speed increases after the second time. For that reason, the end-to-end latency of query executions from the second time onwards is also written in parentheses alongside the latency of the first execution in Table 8.

Because the source code of Seabed is not available, we could not implement it for performance evaluation. However, the source code of order-preserving encryption [33], which is used for *order* operations in Seabed, is available. Consequently, the results of *order* operations executed in the same machine as ours are reported in Table 8. Because the latencies of the *equality* and *join* operations are not available, they are omitted from our comparison.

Because the source code of Arx is unavailable, obtaining the end-to-end latency is challenging. However, the speed of a single predicate processing is reported in [13]. We estimated the latency of 1 M rows, based on the performance of single predicate processing. Note that the machine specification is also different from ours.

Table 8 shows that our system drastically improves query processing speed compared to CryptDB, Arx, and Seabed.

4.4.2 Security evaluation

Lastly, we compare the security level of CryptDB, Arx, Seabed, and our system in Table 9. For *equality* and *join* operations, because Arx and ours both use probabilistic encryption, which encrypts equal values into different ciphertexts, the security level of the two systems can be considered equal. On the other hand, Seabed and CryptDB rely on deterministic encryption, which encrypts data in a predicatable manner, with the result that we consider their security levels to be lower than those of Arx and ours.

For *order* operations, we use ORE, which has a potential weakness in security, where the information of a different block among the four blocks can be leaked. In that sense, our system does not match the security level of Arx, which is reported to allow no leaks. Nevertheless, the ORE we implemented is more secure than the order-preserving encryption [33] in Seabed or the OPE in CryptDB.

We believe that our system is practical in terms of processing speed and achieves a high level of security.

5 Evaluation of TPC-H benchmark

We evaluated the TPC-H benchmark of the proposed system. The TCP-H benchmark has non-sensitive tables such as the order list and line items, as well as a sensitive customer information table. We encrypted the customer information table, except for the customer ID (c_custkey) column, whereas the other tables were stored as plaintext. Thus, we were able to minimize the functional limitations

TABLE 7
Comparison of multi-thread performances with 1 M rows

	CPU w/CC					FPGA w/CC
	1 thread	2 threads	4 threads	8 threads	16 threads	
Equality	500.9 [ms]	297.3 [ms]	182.2 [ms]	97.7 [ms]	74.8 [ms]	8.5 [ms]
Order	199.9 [ms]	125.0 [ms]	73.6 [ms]	48.0 [ms]	45.9 [ms]	24.9 [ms]
Prefix	708.5 [ms]	435.2 [ms]	245.8 [ms]	139.1 [ms]	128.3 [ms]	16.3 [ms]
Substring	4328.6 [ms]	2242.6 [ms]	1201.2 [ms]	707.3 [ms]	595.6 [ms]	76.7 [ms]
Join	5067.7 [ms]	2578.2 [ms]	1339.3 [ms]	759.8 [ms]	647.4 [ms]	83.7 [ms]

TABLE 8
Comparison of latency for 1 M rows with other systems

Scheme	*Equality* [ms]	*order* [ms]	*Join* [ms]
CryptDB	48000 (1670)	46000 (2040)	46000 (2700)
Seabed	-	700	-
Arx	2400	2100000	23680
Ours	**8.5**	**24.9**	**83.7**

TABLE 9
Comparison of security levels

Operation type	Low <- - - - - - - - - - - - - - - - -> High
Equality/Join	CryptDB = Seabed < **Ours** = Arx
Order	CryptDB < Seabed < **Ours** < Arx

resulting from the encryption of the tables while ensuring security.

The experimental environment is the same as that shown in Table 2. We used the TPC-H benchmark tool version 2.18 [49] over the scaling factor of 10 (customer information table is 1.5 M rows). Then, we evaluated the two queries (Q3 and Q5) related to the encrypted customer information table. Because the customer information table was encrypted, original query could not be executed. Therefore, we divided the original query into multiple queries.

Tables 10 and 11 show the original and divided queries in Q3 and Q5, respectively. We evaluated the CPU w/DBC and FPGA w/CC over ciphertext; further we evaluated the original and divided queries over plaintext. In the case of CPU w/DBC and FPGA w/CC, (2) in Table 10 and (2-2) in Table 11 have encrypted query processing because of access to the encrypted customer information table. All the methods were used a single core CPU.

Fig. 14 shows the end-to-end latencies and breakdown performances on Q3 and Q5. The latter was divided by each step, as shown in Tables 10 and 11. There is a very small performance degradation because of the division of the query from the result of the original and divided queries. FPGA w/CC was faster than CPU w/DBC and achieved nearly the same speed as the original query. For the breakdown performance, the encrypted query processing (red bar) in FPGA w/CC can be performed faster than in CPU w/DBC. Thus, we confirmed the effectiveness of the proposed system for real applications.

6 Discussion

Support for large tables. We evaluated tables with up to 10 M rows based on the limitation of 2 GB memory on the FPGA board. To accommodate the larger number of rows, there are two approaches. The first approach is to increase the memory capacity on the FPGA board; we can use an FPGA board that can have up to 256 GB of memory [58] as required. The second approach is to use multiple FPGA boards; it is possible to deal with a large table by storing the divided tables to multiple FPGA boards. The multiple FPGA boards are available over cloud services [59].

Support for multiple search conditions. We evaluated the performance of executing simple queries with only one search condition, but in practice, queries with multiple search conditions are often issued. Our system could support queries with multiple search conditions by first converting such a query into a query plan in which simple queries like *equality* and *order* operations are performed multiple times and then executing these simple queries one by one in the server proxy. However, as the number of search conditions increases, this process must be repeated many times, which in turn slows down the query processing speed. Instead, we could also implement specialized hardware for frequently used search conditions. One significant advantage of using FPGA, which is reprogrammable, is to be able to add specialized hardware for bottleneck operations depending on the workload even after the DBMS is already in operation.

Support for other operations. We implemented four basic query operations in this work. We consider the scalability of our proposed system to be high, because it is designed to be easily modifiable in such cases as changing encryption schemes for the same operation or adding other encryption schemes. However, it is limited to operations that have searchable encryption or execute with combinations of searchable encryption. For instance, *addition* operations on encrypted data can be performed if we implement searchable encryption instead of the SSE and ORE module in Fig. 4 for *addition* operations. As Table 3 shows, the FPGA has sufficient unused resources to enable us to add more functions to the existing accelerator.

Support for queries with no searchable encryption. Our system can support queries for which searchable encryption does not yet exist by working with the application server. For example, *ORDER BY* could be processed by ordering data after it is decrypted on the application server.

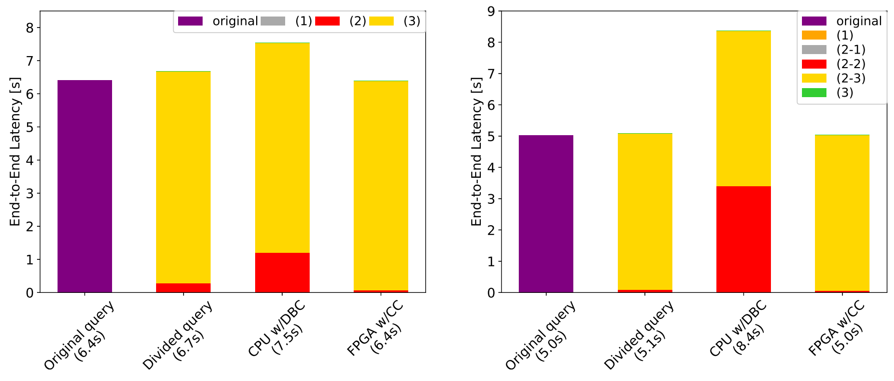
Multi-threaded circuits design. We think that the effects of a speed-up can be further increased if multi-threaded circuits are implemented using free resources in the FPGA. Note that the FPGA requires sufficient memory bandwidth to supply the input data to the multi-threaded circuits for

TABLE 10
Original and divided queries in Q3

Original query		*SELECT* l_orderkey, sum(l_extendedprice * (1 − l_discount)) as revenue, o_orderdate, o_shippriority *FROM* CUSTOMER, ORDERS, LINEITEM *WHERE* c_mktsegment = 'AUTOMOBILE' and c_custkey = o_custkey and l_orderkey = o_orderkey and o_orderdate < date '1995-03-13' and l_shipdate > date '1995-03-13' *GROUP BY* l_orderkey, o_orderdate, o_shippriority *ORDER BY* revenue desc, o_orderdate *LIMIT* 10;
Divided queries	(1)	*CREATE TEMPORARY TABLE* TEMP (t_custkey int not null) ENGINE=MEMORY;
	(2)	*INSERT INTO* TEMP *SELECT* c_custkey *FROM* CUSTOMER *WHERE* c_mktsegment = 'AUTOMOBILE';
	(3)	*SELECT* l_orderkey, sum(l_extendedprice * (1 − l_discount)) as revenue, o_orderdate, o_shippriority *FROM* TEMP, ORDERS, LINEITEM *WHERE* t_custkey = o_custkey and l_orderkey = o_orderkey and o_orderdate < date '1995-03-13' and l_shipdate > date '1995-03-13' *GROUP BY* l_orderkey, o_orderdate, o_shippriority *ORDER BY* revenue desc, o_orderdate *LIMIT* 10;

TABLE 11
Original and divided queries in Q5

Original query		*SELECT* n_name, sum(l_extendedprice * (1 − l_discount)) as revenue from CUSTOMER, ORDERS, LINEITEM, SUPPLIER, NATION, REGION where c_custkey = o_custkey and l_orderkey = o_orderkey and l_suppkey = s_suppkey and c_nationkey = s_nationkey and s_nationkey = n_nationkey and n_regionkey = r_regionkey and r_name = 'MIDDLE EAST' and o_orderdate >= date '1994-01-01' and o_orderdate < date '1994-01-01' + interval '1' year *GROUP BY* n_name *ORDER BY* revenue desc;
Divided queries	(1)	*SELECT* n_name, n_nationkey from NATION, REGION *WHERE* n_regionkey = r_regionkey and r_name = 'MIDDLE EAST';
	(2)	Iterate 2-1) to 2-3) on each n_nationkey$_i$
	(2-1)	*CREATE TEMPORARY TABLE* TEMP$_i$ (t_custkey int not null) ENGINE=MEMORY;
	(2-2)	*INSERT INTO* TEMP$_i$ *SELECT* c_custkey *FROM* CUSTOMER *WHERE* c_nationkey = n_nationkey$_i$;
	(2-3)	*SELECT* sum(l_extendedprice * (1 − l_discount)) as revenue *FROM* TEMP, ORDERS, LINEITEM, SUPPLIER *WHERE* t_custkey = o_custkey and l_orderkey = o_orderkey and l_suppkey = s_suppkey and s_nationkey = n_nationkey$_i$ and o_orderdate >= date '1994-01-01' and o_orderdate < date '1994-01-01' + interval '1' year;
	(3)	Execute "*ORDER BY* revenue desc" and combine n_name



(a) Result of Q3 (b) Result of Q5

Fig. 14. End-to-end latencies and breakdown performances

satisfactory performance.

7 Conclusion

We achieved the acceleration of searchable encrypted DBMS with an FPGA accelerator and Crypto Cache. According to the evaluation using basic queries, the proposed system achieved up to 110.7 times speed-up compared with CPU processing of a single core without Crypto Cache. Moreover, we show that the proposed system can process queries faster than plaintext processing on a CPU when processing large numbers of rows. Furthermore, we confirmed that two queries in the TPC-H benchmark can be executed at almost the same processing speed as that of plaintext processing.

Hence, storing sensitive data safely in a cloud DBMS while minimizing performance degradation is possible with our proposed system. Additionally, the energy efficiency of the proposed FPGA ' s being higher than that of the multi-threaded CPU is also an advantage of using FPGAs in the cloud.

In the future, we plan to implement our system using FPGA instances on cloud services and conduct evaluations.

REFERENCES

[1] Customer data leak deals blow to Benesse [Online], Available: https://asia.nikkei.com/Business/Customer-data-leak-deals-blow-to-Benesse/

[2] Anthem's stolen customer data not encrypted [Online], Available: https://www.cnet.com/news/anthems-hacked-customer-data-was-not-encrypted/

[3] Cyberattack exposes 10M records at Excellus [Online], Available: https://www.computerworld.com/article/2983026/cyberattack-exposes-10m-records-at-excellus.html/

[4] Always encrypted Microsoft [Online], Available: https://docs.microsoft.com/ja-jp/sql/relational-databases/security/encryption/always-encrypted-database-engine/

[5] Oracle DBMS_CRYPTO [Online], Available: https://docs.oracle.com/database/121/ARPLS/d_crypto.htm#ARPLS664/

[6] S. Bajaj and R. Sion, "TrustedDB: A Trusted Hardware based Database with Privacy and Data Confidentiality," in Proc. ACM SIGMOD Conference, pp. 205-216, 2011.

[7] A. Arasu, S. Blanas, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy and R. Venkatesan. "Orthogonal Security With Cipherbase," 6th Biennial Conference on Innovative Data Systems Research (CIDR'13).

[8] A. Gribov, D. Vinayagamurthy and S. Gorbunov, "StealthDB: a Scalable Encrypted Database with Full SQL Query Support," In arXiv:1711.02279 ,2017.

[9] B. Fuhry, R. Bahmani, F. Brasser, F. Hahn, F. Kerschbaum and A. Sadeghi, "HardIDX: Practical and Secure Index with SGX," In arXiv:1703.04583, 2017.

[10] A. Moghimi, G. Irazoqui and T. Eisenbarth, "CacheZoom: How SGX Amplifies The Power of Cache Attacks," In arXiv:1703.06986, 2017.

[11] R. A. Popa, C. M. S. Redfield, N. Zeldovich and H. Balakrishnan, "CryptDB: Protecting Confidentiality with Encrypted Query Processing," Proceedings of the 23rd ACM Symposium on Operating Systems Principles(SOSP '11), pages 85-100, 2011.

[12] A. Papadimitriou1y, R. Bhagwan, N. Chandran, R. Ramjee, A. Haeberleny, H. Singh, A. Modi and S. Badrinarayanan1z, "Big Data Analytics over Encrypted Datasets with Seabed," 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16), 2016.

[13] R. Poddar, T. Boelter and R. A. Popa, "Arx: A Strongly Encrypted Database System," Technical Report No. UCB/EECS-2017-111 [Online], Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-111.html/

[14] Amazon EC2 F1 Instances [Online], Available: https://aws.amazon.com/ec2/instance-types/f1/

[15] IBM cloudFPGA [Online], Available: https://www.zurich.ibm.com/cci/cloudFPGA/

[16] Microsoft Project Catapult Academic Program [Online], Available: https://www.microsoft.com/en-us/research/academic-program/project-catapult-academic-program/

[17] The Netezza FAST Engines Framework [Online], Available: http://www.monash.com/uploads/netezza-fpga.pdf/

[18] C. Dennl, D. Ziener, J. Teich, "On-the-fly Composition of FPGA-Based SQL Query Accelerators Using A Partially Reconfigurable Module Library," IEEE 20th International Symposium on Field-Programmable Custom Computing Machines, pp. 45-52, 2012.

[19] S. Werner, S. Groppe, V. Linnemann and T. Pionteck, "Hardware-accelerated Join Processing in Large Semantic Web Databases with FPGAs", 2013 International Conference on High Performance Computing & Simulation (HPCS), pp.131-138, 2013.

[20] R. J. Halstead, B. Sukhwani, H. Min, M. Thoennes, P. Dube, B. Iyer, S. Asaad, "Accelerating Join Operation for Relational Databases with FPGAs," 21st Annual International IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 17-20, 2013.

[21] A. Becher, D. Ziener, K. M.-Wegener and J. Teich, "A Co-Design Approach for Accelerated SQL Query Processing via FPGA-based Data Filtering," 2015 International Conference on Field Programmable Technology (FPT), 2015.

[22] B. Salami, O. A.-Abella, N. Sonmez, O. Unsal, A. C. Kestelman, "Accelerating Hash-Based Query Processing Operations on FPGAs by a Hash Table Caching Technique," Latin American High Performance Computing Conference(CARLA 2016), pp. 131-145, 2016.

[23] S. Tu, M. F. Kaashoek, S. Madden and N. Zeldovich, "Processing Analytical Queries over Encrypted Data," In Proc. of the 39th international conference on Very Large Data Bases (PVLDB'13), pp. 289-300, 2013.

[24] Y.-F. ZHUANG, C.-Z. WEI, J. LI and W.-G. LI, "Performance Enhanced for CryptDB Based on AES-NI Acceleration," 2017 2nd International Conference on Advances in Management Engineering and Information Technology (AMEIT 2017), pp. 357-361, 2017.

[25] Y. Liu, S. Xue, "Accelerate the Paillier Cryptosystem in CryptDB by Chinese Remainder Theorem,"International Conference on Advanced Communications Technology (ICACT 2018), pp. 74-77, 2018.

[26] S. Dawn, D. Wagner and A. Perrig, "Practical Techniques for Searches on Encrypted Data," Proc. of IEEE Symposium on Security and Privacy (SP), pp. 44-55, 2000.

[27] E.-J. Goh, "Secure indexes," IDIMACS Workshop on Privacy Preserving Data-Mining, 2004.

[28] R. Curtmola, J. Garay, S. Kamara, R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," CCS '06 Proc. of the 13th ACM conference on Computer and communications security, pp. 79-88, 2006.

[29] M. Yoshino, K. Naganuma and H. Satoh, "Symmetric Searchable Encryption for Database Applications," 2011 International Conference on Network-Based Information Systems, pp. 658-662, 2011.

[30] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Ros and M. Steiner, "Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation," NDSS Symposium 2014, 2014.

[31] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," In Proc. of the ACM International Conference on Management of Data, SIGMOD, 2004.

[32] A. Boldyreva, N. Chenettey, Y. Leez, "Order-Preserving Symmetric Encryption," EUROCRYPT 2009, pp. 224 241, 2009.

[33] N. Chenette1, K. Lewi, S. A. Weis and D. J. Wu, "Practical Order-Revealing Encryption with Limited Leakage," In: Peyrin, T. (ed.) FSE 2016. LNCS, vol. 9783, pp. 474-493. Springer, Heidelberg (2016).

[34] K. Lewi, D. J. Wu, "Order-Revealing Encryption: New Constructions, Applications, and Lower Bounds," CCS '16 Proc. of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 1167-1178, 2016.

[35] J. Dyer, M. Dyer, J. Xu, "Order-Preserving Encryption Using Approximate Integer Common Divisors," DPM 2017, CBT 2017: Data Privacy Management, Cryptocurrencies and Blockchain Technology, pp. 257-274, 2017.

[36] D. J. Wu, K. Lewi, "An Implementation of Order-Revealing Encryption," [Online], Available: https://github.com/kevinlewi/fastore/

[37] F. Sky, S. Jarecki, H. Krawczyk, Q. Nguyen, M. Rosu and M. Steiner, "Rich Queries on Encrypted Data: Beyond Exact Matches," Computer Security ESORICS 2015, Lecture Notes in Computer Science, 9327, Springer-Verlag, pp. 123-145, 2015.

[38] C. Melissa and E. Shen, "Substring-Searchable Symmetric Encryption," Proceedings on Privacy Enhancing Technologies, pp. 263-281, 2015.

[39] I. Leontiadis, M. Li, "Storage Efficient Substring Searchable Symmetric Encryption," Cryptology ePrint Archive, Report 2017/153 2017.

[40] F. Hahn, N. Loza, F. Kerschbaum, "Practical and Secure Substring Search," Proc. of the 2018 International Conference on Management of Data, pp. 163-176, 2018.

[41] OmniDB [Online], Available: https://omnidb.org/en/

[42] Postgresql [Online], Available: https://www.postgresql.org/

[43] MySQL [Online], Available: https://www.mysql.com/
[44] Intel FPGA for OpenCL [Online], Available: https://www.intel.com/content/www/us/en/software/programmable/sdk-for-opencl/overview.html/
[45] Faker [Online], Available: https://github.com/joke2k/faker/
[46] S. Duval and G. Leurent, "MDS Matrices with Lightweight Circuits,", Available: https://eprint.iacr.org/2018/260.pdf
[47] CryptDB source code [Online], Available: https://github.com/CryptDB/
[48] mpz_tstbit manual [Online], Available: https://gmplib.org/manual/Integer-Logic-and-Bit-Fiddling.html
[49] TPC-H [Online], Available: http://www.tpc.org/tpch/
[50] V. Migliore, C. Seguin, M. M. Real, V. Lapotre, A. Tisserand, C. Fontaine and G. Gogniat, "A High-Speed Accelerator for Homomorphic Encryption using the Karatsuba Algorithm," ACM Transactions on Embedded Computing Systems, Vol. 16, No. 5, Article 138, 2017.
[51] A. Cilardo and D. Argenziano, "Securing the Cloud with Reconfigurable Computing: An FPGA Accelerator for Homomorphic Encryption," 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1622-1627, 2016.
[52] F. Boucenna, O. Nouali, A. Dabah and S. Kechid, "Accelerated Search over Encrypted Cloud Data," 2017 IEEE International Conference on Big Data and Smart Computing (BigComp), pp. 170-177, 2017.
[53] Z. Wang, J. Paul, Hui Y. Cheah, B. He and W. Zhang, "Relational Query Processing on OpenCL-based FPGAs," 2016 26th International Conference on Field Programmable Logic and Applications (FPL), 2016.
[54] B. Salami, G. A. Malazgirt, O. A.-Abella, A. Yurdakul and N. Sonmez, "AxleDB: A novel programmable query processing platform on FPGA," Microprocessors and Microsystems Vol. 51, pp. 142-164, 2017.
[55] S.-W. Jun, M. Liu, S. Lee, J. Hicks, J. Ankcorn, M. King, S. Xu and Arvind, "Bluedbm: An appliance for big data analytics," 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA), 2015.
[56] L. Woods, Z. István and G. Alonso, "Ibex: an intelligent storage engine with support for advanced SQL offloading," Proceedings of the VLDB Endowment Vol. 7, Issue 11, pp. 963-974, 2014.
[57] Early Power Estimators and Power Analyzer [Online], Available: https://www.intel.com/content/www/us/en/programmable/support/support-resources/operation-and-testing/power/pow-powerplay.html
[58] 520N-MX PCIe FPGA Board [Online], Available: https://www.bittware.com/wp-content/uploads/datasheets/ds-520n-mx.pdf
[59] Amazon EC2 F1 Instances [Online], Available: https://aws.amazon.com/ec2/instance-types/f1

PLACE PHOTO HERE

Mitsuhiro Okada received the MS degree in electrical engineering from Tokyo University of Science, Japan, in 2006. He is currently a researcher at Hitachi Ltd., with Research & Development Group. His research interests include reconfigurable computing, high-performance computing, human activity recognition and image compression.

PLACE PHOTO HERE

Takayuki Suzuki received the ME degree in electrical engineering and Information Science from Osaka Prefecture University, Japan, in 2002. He is currently an engineer at Hitachi Ltd., with Financial Institutions Business Unit. His research interests include embedded systems, software development, computer security and blockchain-related technology.

PLACE PHOTO HERE

Naoya Nishio received the MS degree in informatics from Kyoto University, Japan, in 2015. He is currently a researcher at Hitachi Ltd., with Research & Development Group. His research interests include machine learning, multi-agent simulation, database management system and cognitive science.

PLACE PHOTO HERE

Hasitha Muthumala Waidyasooriya received the B.E. degree in information engineering, the M.S. degree in information sciences, and the Ph.D. degree in information sciences from Tohoku University, Japan, in 2006, 2008, and 2010, respectively, where he is currently an Assistant Professor with the Graduate School of Information Sciences. His research interests include reconfigurable computing, processor architectures for big-data processing, and high-level design methodology for VLSIs.

PLACE PHOTO HERE

Masanori Hariyama received the B.E. degree in electronic engineering, the M.S. degree in information sciences, and the Ph.D. degree in information sciences from Tohoku University, Sendai, Japan, in 1992, 1994, and 1997, respectively, where he is currently a Professor with the Graduate School of Information Sciences. His research interests include real-world applications such as robotics and medical applications, big data applications such as bio-informatics, high-performance computing, VLSI computing for real-world application, high-level design methodology for VLSIs, and reconfigurable computing.