Cryptanalysis of An Encrypted Database in SIGMOD '14

Xinle Cao
Zhejiang University
Hangzhou, China
xinle@zju.edu.cn

Jian Liu*
Zhejiang University
Hangzhou, China
liujian2411@zju.edu.cn

Hao Lu
Zhejiang University
Hangzhou, China
luhao@zju.edu.cn

Kui Ren
Zhejiang University
Hangzhou, China
kuiren@zju.edu.cn

ABSTRACT

Encrypted database is an innovative technology proposed to solve the data confidentiality issue in cloud-based DB systems. It allows a data owner to encrypt its database before uploading it to the service provider; and it allows the service provider to execute SQL queries over the encrypted data. Most of existing encrypted databases (e.g., CryptDB in SOSP '11) do not support data interoperability: unable to process complex queries that require piping the output of one operation to another.

To the best of our knowledge, SDB (SIGMOD '14) is the only encrypted database that achieves data interoperability. Unfortunately, we found SDB is not secure! In this paper, we revisit the security of SDB and propose a ciphertext-only attack named *co-prime attack*. It successfully attacks the common operations supported by SDB, including *addition*, *comparison*, *sum*, *equi-join* and *group-by*. We evaluate our attack in three real-world benchmarks. For columns that support *addition* and *comparison*, we recover 84.9% – 99.9% plaintexts. For columns that support *sum*, *equi-join* and *group-by*, we recover 100% plaintexts.

Besides, we provide potential countermeasures that can prevent the attacks against *sum*, *equi-join*, *group-by* and *addition*. It is still an open problem to prevent the attack against *comparison*.

PVLDB Reference Format:
Xinle Cao, Jian Liu, Hao Lu, and Kui Ren. Cryptanalysis of An Encrypted Database in SIGMOD '14. PVLDB, 14(10): 1743 - 1755, 2021.
doi:10.14778/3467861.3467865

1 INTRODUCTION

Database-as-a-service (DBaaS) is a prevalent cloud-service paradigm allowing a data owner (DO) to outsource its database to a service provider (SP) that possesses high-performance machines and sophisticated database software. DO can query the database as if it was stored locally. SP thus provides storage, computation and administration services to DO. Most importantly, this paradigm provides elasticity to DO: they can scale their service consumption up or down according to their real requirements.

On the other hand, this service paradigm brings data confidentiality issue to DO, as the database stored in SP as well as the queries sent by DO may contain sensitive information. A hacker can exploit software vulnerabilities to break into the server and snoop on the data [2] and curious administrators of SP can steal data they are interested in [1]. One approach to prevent this potential information leakage is to encrypt sensitive data before storing it at SP, e.g., Depot [18], SUNDR [17] and SPORC [9]. To process queries, the encrypted data has to be shipped back to DO and processed locally. Unfortunately, for some operations, this approach incurs a huge communication overhead that is in the order of the database size, hence it is even worse than storing the database locally.

To this end, Popa et al. [22] propose the first *encrypted database* called CryptDB allowing SP to execute SQL queries directly over encrypted data. The core idea of CryptDB is to encrypt each data item in one or more *onions*: different onions enable different kinds of operations; within each onion, an item is dressed in layers of increasingly stronger encryption. For example, it uses *homomorphic encryption* (HE) [21] for addition in one onion, and uses *order-preserving encryption* (OPE) [3, 6] for comparison in another onion. However, CryptDB is unable to process complex queries that require piping the output of one operation (onion) to another (i.e., *data interoperability*). For example, the selection clause "$quantity \times unit\text{-}price > \$10,000$" that requires both multiplication and comparison at the same time cannot be processed by CryptDB.

SDB in SIGMOD '14. To achieve data interoperability, in SIGMOD '14, Wong et al. presented an encrypted database named SDB [26]. It uses a special kind of multiplicatively homomorphic encryption scheme to encrypt each data item: when ciphertexts under different keys being multiplied with each other, it generates a new key:

$$E(k_3, v_1 \times v_2) \leftarrow E(k_1, v_1) \times E(k_2, v_2)$$

Besides, it is additively homomorphic only for ciphertexts under the same key:

$$E(k, v_1 + v_2) \leftarrow E(k, v_1) + E(k, v_2)$$

When a database is stored, all data items are encrypted under different keys. When DO wants to add two columns, SDB provides a *KeyUpdate* operation, which enables SP (with the assistance of DO) to update the items in the same row of these two columns to be under the same key, so that addition can be done. The *KeyUpdate* operation also allows SP and DO to decrypt a whole column with constant communication overhead.

*Jian Liu is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 14, No. 10 ISSN 2150-8097.
doi:10.14778/3467861.3467865

To do *comparison*, e.g., $v_1 - v_2 > 0$, SP (with the assistance of DO) first updates $E(k_1, v_1)$ and $E(k_2, v_2)$ to be under the same key k_3, and computes $E(k_3, v_1 - v_2)$. Then, SP computes

$$E(k_5, u(v_1 - v_2)) \leftarrow E(k_3, v_1 - v_2) \times E(k_4, u),$$

where $E(k_4, u)$ was previously uploaded to SP by DO, and u is a small random number that will not change the sign of $(v_1 - v_2)$. In the end, SP and DO decrypt the whole column using *KeyUpdate* so that SP can return the rows that satisfy $u(v_1 - v_2) > 0$.

To *sum* a column of n items $E(k_1, v_1), ..., E(k_n, v_n)$, SP (with the assistance of DO) updates these encrypted items to $m^{-1}v_1, ..., m^{-1}v_n$, where m is a new random number. Then, SP returns $m^{-1} \sum_{i=1}^{n} v_i$ to DO. The *equi-join* and *group-by* operations can be realized in the same way as *sum*.

Our contribution. In this paper, we revisit SDB and make the following observations:

(1) It cannot encrypt 0s;
(2) The ciphertexts for *addition* are deterministic: $E(k, v_1) = E(k, v_2)$ iff $v_1 = v_2$;
(3) If the ciphertexts $E(k_1, v_1), ..., E(k_t, v_t)$ have been updated to be under the same key k_0, then these ciphertexts can be converted back to be under any key within $\{k_0, k_1, ..., k_t\}$.
(4) The decryption procedure in *comparison* not only applies to the difference (i.e., $v_1 - v_2$), but also applies to the items being compared (i.e., v_1 and v_2).

Based on these observations, we present a ciphertext-only attack against five operations in SDB: *addition*, *sum*, *comparison*, *equi-join* and *group-by*. We name it *co-prime attack*, because its success rate highly depends on the theorem [16, 20]:

given α random positive integers in $\mathbb{Z}_M$, the probability for them to be co-prime is $\frac{1}{\zeta(\alpha)} + O(\frac{1}{|M|})$,

where ζ refers to the Riemann ζ-function and $\zeta(\alpha) = \sum_{i=1}^{+\infty} \frac{1}{i^\alpha}$. This probability is close to 92.4% when $\alpha = 4$, and close to 99.9% when $\alpha = 10$.

Below, we briefly explain the co-prime attack against *addition*, *sum*, *equi-join*, *group-by* and *comparison*:

- **Addition.** To attack *addition*, SP calculates $E(k, v_1) : E(k, v_2) = \gamma_1 : \gamma_2$; if v_1 and v_2 are co-prime, then $v_1 = \gamma_1$ and $v_2 = \gamma_2$. With more items being added together, the co-prime probability of these elements increases. Our experimental results show that we can recover at least 90% plaintexts with 7 columns added together (cf. Table 6).
- **Sum, equi-join and group-by.** In the *sum* operation, SP gets $m^{-1}v_1, ..., m^{-1}v_n$. Similar to *addition*, SP can get ratio among all elements in this column. In our experiments, with the number of rows being more than 100, we can recover all plaintexts of this column with almost 100% probability. This attack applies to *equi-join* and *group-by* operations as well.
- **Comparison.** Recall that after *comparison* between v_1 and v_2, $u(v_1 - v_2)$ is revealed to SP. Based on the 4-th observation aforementioned, both uv_1 and uv_2 will be revealed to SP as well. If v_1 has been compared for twice, uv_1 and $u'v_1$ will be revealed. If u and u' are co-prime, v_1 will be revealed. In our experiments, if the *comparison* operation has been applied to a column for 4 times, we can recover at least 90% plaintexts of this column.

We summarize our contribution as follows:

(1) We revisit SDB (SIGMOD '14) and make four observations, which incur serious information leakage but was not mentioned in their paper (Section 2).
(2) We propose a ciphertext-only attack (named *co-prime attack*) against the *addition*, *sum*, *comparison*, *equi-join* and *group-by* operations in SDB (Section 4).
(3) We validate our attack on three public benchmarks, UCI Credit Card Clients, TPC-C and TPC-H (Section 5). The experimental results are summarized in Table 1.
(4) We provide potential countermeasures that can prevent the attacks against *sum*, *equi-join*, *group-by* and *addition* (Section 6). It is still an open problem to prevent the attacks against *comparison*.

Table 1: Summary of our experimental results. The recovery rates are for the columns that are executed with the operations. More details are in Section 5.

Operation	Benchmark	Recovery (%)	Requirements
Addition	Credit	97.3	1 addition query
	TPC-C	84.9	4 update queries
Comparison	Credit	99.8	10 range queries
	TPC-C	99.8	
	TPC-H	99.9	
Sum	Credit	100	1 sum query
	TPC-C	100	
	TPC-H	100	
Equi-Join	Credit	100	1 equi-join query
	TPC-C	100	
	TPC-H	100	
Group-by	Credit	100	1 group-by query
	TPC-C	100	
	TPC-H	100	

2 SDB REVISIT

In this section, we revisit the security of SDB. We first provide technical details of SDB and summarize them into interfaces, so that we can use these interfaces to explain our attack. Then, we make some observations, which result in serious information leakage but was not mentioned in their paper.

2.1 SDB in detail

SDB was proposed in SIGMOD '14, by Wong et al. [26], to address the data interoperability issue in encrypted databases. It mainly consists of the following operations:

Setup. DO generates two big random prime numbers ρ_1 and ρ_2; calculates $N = \rho_1\rho_2$ and $\phi(N) = (\rho_1 - 1)(\rho_2 - 1)$; and samples

a generator $g \in \mathbb{Z}_N$ for $\phi(N)$ order group. For each column, DO generates a column key $ck = \langle m, x \rangle$, where $m \in \mathbb{Z}_N$ and $x \in \mathbb{Z}_{\phi(N)}$; and for each row, DO generates a row key $r \in \mathbb{Z}_{\phi(N)}$. Then, the encryption key for each data item is $k = \langle \langle m, x \rangle, r \rangle$.

$$\boxed{(N, g, k_1, k_2, ...) \leftarrow Setup(1^\lambda)}$$

Encryption. DO takes the encryption key $k = \langle \langle m, x \rangle, r \rangle$ and a data item v, computes the ciphertext

$$c := (mg^{rx})^{-1} \cdot v \mod N$$

.

$$\boxed{c \leftarrow E(k, v)}$$

Multiplication. To multiply two columns, SP can directly multiply the two data items in the same row:

$$(m_3 g^{rx_3})^{-1} v_1 v_2 := (m_1 g^{rx_1})^{-1} v_1 \cdot (m_2 g^{rx_2})^{-1} v_2 \mod N.$$

Notice that $k_3 = \langle \langle m_3, x_3 \rangle, r \rangle$ is the new-generated encryption key for the product $v_1 v_2$, where $m_3 = m_1 \cdot m_2 \mod N$ and $x_3 = x_1 + x_2 \mod \phi(N)$.

$$\boxed{E(k_3, v_1 v_2) \leftarrow E(k_1, v_1) \times E(k_2, v_2)}$$

Key update. The *KeyUpdate* operation allows SP (with the assistance of DO) to update the column key of a ciphertext to a new one. Suppose DO wants to update the encryption key $k_1 = \langle \langle m_1, x_1 \rangle, r \rangle$ of a ciphertext $E(k_1, v)$ to $k_2 = \langle \langle m_2, x_2 \rangle, r \rangle$; and there is a previously uploaded column of ciphertexts of 1 which includes a ciphertext encrypted under $k_3 = \langle \langle m_3, x_3 \rangle, r \rangle$.

There are two phases in *KeyUpdate*:

(1) In the first phase, DO computes

$$p = x_3^{-1}(x_2 - x_1) \mod \phi(N)$$

$$q = m_1 m_3^p m_2^{-1} \mod N,$$

and includes them in the query Q being sent to SP.

(2) In the second phase, SP computes

$$E(k_2, v) = q \cdot E(k_1, v) \cdot E(k_3, 1)^p \mod N.$$

Notice that p and q can be applied to the whole column of $E(k_1, v)$.

Notice that all *KeyUpdate* operations are issued by DO, SP can only passively execute these operations and never ask for additional *KeyUpdate* operations.

$$\boxed{E(k_2, v) \leftarrow KeyUpdate(E(k_1, v), Q)}$$

Addition. To add two columns, SP first (with the assistance of DO) updates the column keys to the same one using *KeyUpdate*. Then, the two items in the same row would be: $c_1 = (m_3 g^{rx_3})^{-1} \cdot v_1$ and $c_2 = (m_3 g^{rx_3})^{-1} \cdot v_2$. SP can add them directly:

$$(m_3 g^{rx_3})^{-1}(v_1 + v_2) := (m_3 g^{rx_3})^{-1} v_1 + (m_3 g^{rx_3})^{-1} v_2 \mod N.$$

$$\boxed{E(k_3, v_1 + v_2) \leftarrow E(k_1, v_1) + E(k_2, v_2)}$$

Sum. To sum all items in one column, SP (with the assistance of DO) updates the column key to $ck = \langle m, 0 \rangle$. Then, the ciphertexts in this column become: $m^{-1}v_1, ..., m^{-1}v_n$. SP simply returns $m^{-1} \sum_{i=1}^{n} v_i \mod N$.

$$\boxed{m^{-1} \sum_{i=1}^{n} v_i \leftarrow sum(E(k_1, v_1), ..., E(k_n, v_n))}$$

Comparison. The *comparison* operation frequently appears in range queries with a selection clause. Given two encrypted columns V_1 and V_2, the object is to compare the items (v_1 and v_2) in each row and return the rows that meet the condition. Notice that the comparison results should be visible to SP so that it can decide which rows to return.

To compare two values $E(k_1, v_1)$ and $E(k_2, v_2)$) in the same row, SP (with the assistance of DO) first updates them to be under the same key k_3 and computes $E(k_3, v_1 - v_2)$. Then, SP computes

$$E(k_5, u(v_1 - v_2)) \leftarrow E(k_3, v_1 - v_2) \times E(k_4, u),$$

where $E(k_4, u)$ was previously uploaded to SP by DO, and u is a small random number that will not change the sign of $(v_1 - v_2)$. In the end, SP (with the assistance of DO) updates the column key of $E(k_5, u(v_1 - v_2))$ to $\langle 1, 0 \rangle$, which gives $u(v_1 - v_2)$ to SP. Then, SP can decide the truth value of the comparison by comparing $u(v_1 - v_2)$ with $\frac{N}{2}$.

$$\boxed{u(v_1 - v_2) \leftarrow cmp(E(k_1, v_1), E(k_2, v_2))}$$

Equi-Join. The *equi-join* operation is used to combine rows from two or more tables, through a common column between them. To join two encrypted tables T_1 and T_2 in SDB, SP (with the assistance of DO) updates the column keys of the corresponding columns to $ck = \langle m, 0 \rangle$. Then, the ciphertexts in these two columns are $m^{-1}v_1, m^{-1}v_2, ...$ and $m^{-1}v_1', m^{-1}v_2', ...$ It is clear that the same items result in the same ciphertexts so that SP can join the tables.

$$\boxed{T_3 \leftarrow equi\text{-}join(T_1, T_2)}$$

Group-by. The *group-by* operation groups rows that have the same values into summary rows. To group-by items in one column, SP (with the assistance of DO) updates the column key to $ck = \langle m, 0 \rangle$. Then, the ciphertexts in this column become: $m^{-1}v_1, ..., m^{-1}v_n$. As a result, the same items result in the same ciphertexts so that SP can do group-by.

$$\boxed{groups \leftarrow group\text{-}by(E(k_1, v_1), ..., E(k_n, v_n))}$$

2.2 Observations

Intuitively, SDB is a special kind of multiplicatively homomorphic encryption scheme: when ciphertexts under different keys being multiplied with each other, it generates a new key; and it supports key update. However, it uses some tricks to support addition and comparison, which introduces some security flaws. We make the following observations on their security flaws.

Observation 1. SDB *cannot encrypt* 0*s. Specifically, in* SDB*, the ciphertext c = 0 iff the plaintext v = 0.*

Proof. It is obvious that if $v = 0$, then

$$c = (mg^{rx})^{-1} \cdot v \mod N = 0.$$

For the opposite direction, if $c = 0$ but $v \neq 0$, then $(mg^{rx})^{-1} \cdot v$ *must be* a multiple of $N = \rho_1 \rho_2$, where ρ_1 and ρ_2 are two big primes. We prove this will never happen.

Notice that v is a plaintext smaller than any of ρ_1 and ρ_2. Then, if $(mg^{rx})^{-1} \cdot v$ is a multiple of $N = \rho_1 \rho_2$, $(mg^{rx})^{-1}$ must be a multiple of N. However, this is not true as $(mg^{rx})^{-1} \in \mathbb{Z}_N$. Therefore, $(mg^{rx})^{-1} \cdot v$ will never be a multiple of N, hence v must be 0. □

Observation 2. *The ciphertexts for addition are deterministic:* $E(k, v_1) = E(k, v_2)$ *iff* $v_1 = v_2$*;*

Proof. It is obvious that if $v_1 = v_2$, then

$$(mg^{rx})^{-1}v_1 \mod N = (mg^{rx})^{-1}v_2 \mod N$$

For the opposite direction, if $E(k, v_1) = E(k, v_2)$, then

$$(mg^{rx})^{-1}(v_1 - v_2) \equiv 0 \mod N$$

Based on Observation 1, we have $v_1 - v_2 = 0$, then $v_1 = v_2$. □

Observation 3. *If* $E(k_1, v_1), ..., E(k_t, v_t)$ *in the same row but different columns have been updated to be under the same key* k_0 *via KeyUpdate, then these ciphertexts can be converted back to be under any key within* $\{k_0, k_1, ..., k_t\}$.

Proof. As assumed in this observation, there exists a query Q_i that can update $E(k_i, v_i)$ to be under k_0.

$$E(k_0, v_i) \leftarrow KeyUpdate(E(k_i, v_i), Q_i)$$

As we describe in section 2.1, the update process is computed as:

$$E(k_0, v_i) = q_i \cdot E(k_i, v_i) \cdot E(k', 1)^{p_i} \mod N$$

It is clear that this equation is reversible:

$$E(k_i, v_i) = q_i^{-1} \cdot E(k_0, v_i) \cdot (E(k', 1)^{p_i})^{-1} \mod N$$

To prove that $E(k_i, v_i)$ can be converted to any $E(k_j, v_i)$ with $(j \neq i)$, we only need to prove that $E(k_0, v_i)$ can be converted to $E(k_j, v_i)$ with $(j \neq i)$. This can be achieved by:

$$E(k_j, v_i) = q_j^{-1} \cdot E(k_0, v_i) \cdot (E(k', 1)^{p_j})^{-1} \mod N.$$

□

Since q_i and p_i can be applied to the whole column, the following statement also holds.

Remark 1. *If columns* $V_1, ..., V_t$ *(each encrypted under a column key* ck_i*) can be converted to be under the same column key* ck_0 *via KeyUpdate, then these columns can be converted back to any column key within* $\{ck_0, ck_1, ..., ck_t\}$.

Observation 4. *The decryption procedure in comparison not only applies to the difference, but also applies to the items being compared. Specifically, if DO issues a query*

$$u(v_1 - v_2) \leftarrow cmp(E(k_1, v_1), E(k_2, v_2)),$$

then SP can get both uv_1 *and* uv_2.

Proof. Recall that the *comparison* operation is executed as following steps:

(1) $E(k_3, v_1) \leftarrow KeyUpdate(E(k_1, v_1), Q)$;
(2) $E(k_3, v_2) \leftarrow KeyUpdate(E(k_2, v_2), Q)$;
(3) $E(k_3, v_1 - v_2) = E(k_3, v_1) - E(k_3, v_2)$;
(4) $E(k_5, u(v_1 - v_2)) \leftarrow E(k_3, v_1 - v_2) \times E(k_4, u)$;
(5) $u(v_1 - v_2) \leftarrow KeyUpdate(E(k_5, u(v_1 - v_2)), Q)$

In Step 4, SP can use $E(k_3, v_1)$ to substitute $E(k_3, v_1 - v_2)$, so that it can get uv_1 in the end. Similarly, it can get uv_2. □

Clearly, these features we observed lead to serious information leakage. We will explain how we exploit these leakage to develop our attacks in Section 4. Furthermore, our attacks are independent of the size of the security parameters, as these observations hold for any size of the security parameters.

3 ADVERSARIAL MODEL

We only assume SP is an honest-but-curious adversary, i.e., it will not deviate from the protocol specification, but will attempt to learn all possible information from legitimately received messages (or intermediate results). We assume the adversary has access to the encrypted database but cannot issue queries; the adversary has no auxiliary information (e.g., application details, public statistics or prior versions) about the database; and known-plaintext attack is not available to the adversary. In another words, ciphertext-only attack is the only option for the adversary. We remark that this adversarial model is much weaker than what is typically assumed in attack-related literature for encrypted databases (cf. Section 8). We further emphasis that this adversarial model captures all threats that database customers are typically worried about: internal threats like curious database administrators or employees, and external threats like individual hackers or organized crime.

We introduce new notations as needed. A summary of notations appears in Table 2.

Table 2: Summary of notations.

Notation	Description
SP	service provider
DO	data owner
M	plaintext space
N	ciphertext space
Q	a query
v	value of a data item
V	a column of values
c	ciphertext
k	encryption key: $k = \langle \langle m, x \rangle, r \rangle$
$E()$	encryption function
ρ	prime
α	number of co-prime integers
n	number of rows
ck	column key
r	row key

4 CO-PRIME ATTACK AGAINST SDB

In this section, we explain how we exploit the observations we made in Section 2.2 to develop our attacks against SDB.

4.1 Co-prime probability

We first present the following theorem (proved in [16, 20]) about co-prime probability, which is considered to be the theoretical foundation for our attacks.

Theorem 1. *Given α positive integers that are chosen uniformly at random from $\mathbb{Z}_M$, the probability that they are co-prime is:*

$$\frac{1}{\zeta(\alpha)} + O(\frac{1}{|M|}),$$

where ζ refers to the Riemann ζ-function and $\zeta(\alpha) = \sum_{i=1}^{+\infty} \frac{1}{i^\alpha}$.

Notice that $O(\frac{1}{|M|})$ can be ignored when $|M|$ is large, then the co-prime probability becomes $\frac{1}{\zeta(\alpha)}$. Figure 1 shows $\frac{1}{\zeta(\alpha)}$ for α random integers. For instance, the probability for 4 randomly chosen integers being co-prime is nearly 92.4%; the probability for 10 randomly chosen integers being co-prime is close to 99.9%. To estimate the error introduced by ignoring $O(\frac{1}{|M|})$, we set M to 2^{24} and 2^{80} respectively, pick 10^{10} sets of α random numbers, and calculate the proportion of sets whose numbers are co-prime. Figure 2 shows the difference between $\frac{1}{\zeta(\alpha)}$ and the experimentally calculated co-prime probabilities.

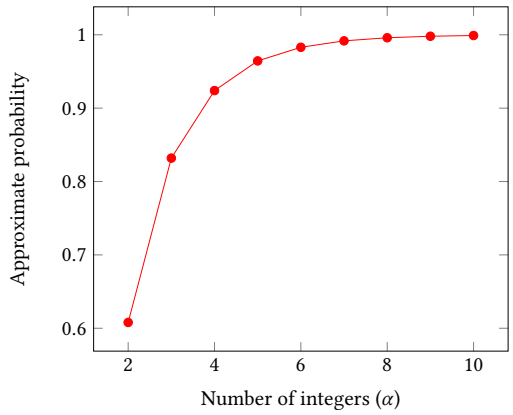


Figure 1: Approximate probability for different number of integers being co-prime.

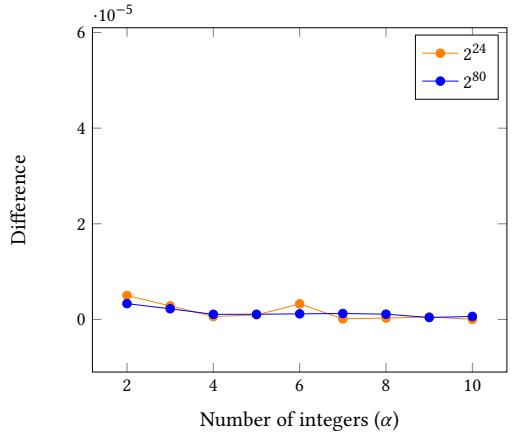


Figure 2: Difference between the experimentally calculated probabilities and $\frac{1}{\zeta(\alpha)}$ for different number of integers.

Figure 1 and Figure 2 show the co-prime probability of random numbers in finite fields are high. For instance, the probability for 4 randomly chosen integers being co-prime is ~ 92.4%; the probability for 10 randomly chosen integers being co-prime is ~ 99.9%. The success rate of our attack highly depends on this co-prime probability, hence we name our attack *co-prime attack*. It utilizes *either* of the following information that is revealed by SDB:

- **Ratio among different plaintexts (revealed by *addition or sum/equi-join/group*)**. Once the ratio among α plaintexts is revealed, all these α plaintexts will be revealed if they are co-prime. We assume, w.h.p. these α plaintexts are co-prime (our experimental results confirm this assumption).
- **Products of the same plaintext with different coefficients (revealed by comparison)**. The coefficients are random numbers used to mask the plaintext, thereby, w.h.p. they are co-prime. Then, the greatest common divisor of these products is exactly the plaintext.

In the rest of this section, we explain how we exploit the legitimate queries to cause the above information leakage.

4.2 Co-prime attack against addition

Algorithm 1 shows the co-prime attack against the *addition* operation in SDB. It takes α ciphertexts in the same row together with a query calculating the sum of the corresponding plaintexts, and it recovers all these α plaintexts. **Notice that a query typically operates on the whole columns, hence we can recover all ciphertexts in the whole α columns by running Algorithm 1 for each row separately.**

As we described in Section 2.1, to answer a query Q that adds α items in a row, SP (with the assistance of DO) executes the following two steps:

(1) Receiving respective Q from DO, SP executes the second phase of *KeyUpdate* to convert all α ciphertexts to be under the same key:

$$E(k_0, v_i) \leftarrow KeyUpdate(E(k_i, v_i), Q) \quad i \in [\alpha].$$

(2) With all ciphertexts under the same key, SP can execute *addition* directly:

$$E(k_0, \sum_{i=1}^{\alpha} v_i) \leftarrow \sum_{i=1}^{\alpha} E(k_0, v_i).$$

After the first step above (line 1-3), SP computes the ratio between v_1 and other plaintexts (line 4-12). This is possible because after *KeyUpdate* converts ciphertexts to be under the same key, the ciphertexts become both deterministic and additively homomorphic:

- deterministic: if $v = v'$ then $E(k_0, v) = E(k_0, v')$;
- additively homomorphic: $\gamma E(k_0, v) = E(k_0, \gamma v)$, which can be considered as adding γ ciphertexts together.

With these two properties, SP can compute the ratio between v_1 and any v_i, if they are non-zero (cf. Observation 1), by finding $\gamma_1^{(i)}$ and γ_i s.t.,

$$\gamma_i E(k_0, v_1) = \gamma_1^{(i)} E(k_0, v_i).$$

Then $v_1 : v_i = \gamma_1^{(i)} : \gamma_i$. SP can find $\left\langle \gamma_1^{(i)}, \gamma_i \right\rangle$ by trying each pair in $\mathbb{Z}_M^2$, which introduces $O(M^2)$ computational complexity and $O(1)$

Algorithm 1: Co-prime attack against *addition* (for 1 row)

Input: $\{E(k_1, v_1), ..., E(k_\alpha, v_\alpha)\}$; a query Q with an operation $v_1 + ... + v_\alpha$

Output: $v_1, ..., v_\alpha$

1 **for** $i = 1 \rightarrow \alpha$ **do** update to the same key
2 $E(k_0, v_i) \leftarrow KeyUpdate(E(k_i, v_i), Q)$ with the assistance of DO
3 **end**
4 **for** $i = 2 \rightarrow \alpha$ **do** compute the ratio between v_1 and v_i
5 **for** $\gamma_1^{(i)} = 1 \rightarrow M$ **do**
6 **for** $\gamma_i = 1 \rightarrow M$ **do**
7 **if** $(\gamma_i E(k_0, v_1) \mod N) = (\gamma_1^{(i)} E(k_0, v_i) \mod N)$ **then**
8 **go to** line 4
9 **end**
10 **end**
11 **end**
12 **end**
13 $v_1 \leftarrow \text{SmallestCommonMultiple}(\gamma_1^{(2)}, ..., \gamma_1^{(\alpha)})$
14 **for** $i = 2 \rightarrow \alpha$ **do**
15 $v_i := v_1 \cdot \frac{\gamma_i}{\gamma_1^{(i)}}$
16 **end**
17 **return:** $v_1, ..., v_\alpha$

space complexity. We have a way to reduce $O(M^2)$ computational complexity to $O(M)$, but $O(M)$ space complexity is required. More specifically, we maintain a key-value store; in the first loop, we put each $(\gamma_1^{(i)} \cdot E(k_0, v_i), \gamma_1^{(i)})$ into the store; in the second loop, whenever we find $\gamma_i \cdot E(k_0, v_1)$ is in the store, we know γ_i and $\gamma_1^{(i)}$ are the values we want.

When some of the vs are negative, to find the ratio between v_1 and v_i, we run line 5-11 for two times: in the first time, we assume they are with the same sign and run it as before; if it fails, in the second time, we assume they are with opposite sign and multiply -1 to $E(k_0, v_i)$, and then run as before[1].

After finding all ratios: $\left\langle \gamma_1^{(2)}, \gamma_2 \right\rangle, ..., \left\langle \gamma_1^{(\alpha)}, \gamma_\alpha \right\rangle$, SP can merge them by calculating the smallest common multiple of $\left\langle \gamma_1^{(2)}, ..., \gamma_1^{(\alpha)} \right\rangle$:

$$\gamma_1 \leftarrow \text{SmallestCommonMultiple}(\gamma_1^{(2)}, ..., \gamma_1^{(\alpha)}).$$

Then, the ratio among $v_1, ..., v_\alpha$ is:

$$\gamma_1 : \frac{\gamma_1\gamma_2}{\gamma_1^{(2)}} : ... : \frac{\gamma_1\gamma_\alpha}{\gamma_1^{(\alpha)}}$$

If $v_1, ..., v_\alpha$ are co-prime, SP can recover all of them:

$$v_1 := \gamma_1$$
$$v_2 := \frac{\gamma_1\gamma_2}{\gamma_1^{(2)}}$$
$$...$$
$$v_\alpha := \frac{\gamma_1\gamma_\alpha}{\gamma_1^{(\alpha)}}.$$

Recall that the co-prime probability for $v_1, ..., v_\alpha$ is high when α is large. Even for $\alpha = 4$, the co-prime probability is still $\sim 92.4\%$.

Notice that Algorithm 1 only considers a single query that adds all α columns. Next, we show that SP can still recover the plaintexts even if these α columns are queried separately as long as the queries are *connected*.

Definition 1 (Connected queries). *Suppose query Q involves addition among a set of columns $\{V_1, V_2, ...\}$, and query Q' involves addition among another set of columns $\{V_1', V_2', ...\}$. Then, Q and Q' are connected if $\{V_1, V_2, ...\} \cap \{V_1', V_2', ...\} \neq \emptyset$. We say a set of queries are connected if any two of them can be connected with each other.*

Observation 5. *If a set of queries are connected, the columns touched by these queries can be converted to be under any column key of these columns.*

Proof. We first prove that this statement holds for two connected queries Q and Q': $\{V_1, V_2, ...\}$ and $\{V_1', V_2', ...\}$ are touched by Q and Q' separately. Suppose there is a column $V_d \in \{V_1, V_2, ...\} \cap \{V_1', V_2', ...\}$ and its column key is ck_d. According to Remark 1, columns in $\{V_1, V_2, ...\}$ can be converted to be under the same column key $ck_d \in \{ck_1, ...\}$; Similarly, columns in $\{V_1', V_2', ...\}$ can be converted to be under $ck_d \in \{ck_1', ...\}$ as well. As $\{V_1, V_2, ...\} \cup \{V_1', V_2', ...\}$ can be converted to be under the same key ck_d, based on Remark 1 again, columns in $\{V_1, V_2, ...\} \cup \{V_1', V_2', ...\}$ can be converted to be under any key in $\{ck_1, ck_2, ...\} \cup \{ck_1', ck_2', ...\}$.

This can be easily extended to multiple connected queries. □

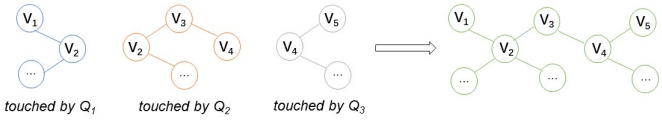


Figure 3: Q_1 and Q_2 are connected queries; Q_2 and Q_3 are connected queries. Then, the columns touched by Q_1, Q_2 and Q_3 can be converted to be under the same column key.

Figure 3 visualizes Observation 5.

Observation 5 implies that a set of connected queries, each of which adds a set of columns, is equal to a single query that adds all these columns. Then, Algorithm 1 can be applied directly.

4.3 Co-prime attacks against sum, equi-join and group-by

The co-prime attack against the *sum* operation in SDB is shown in Algorithm 2. It takes a column of ciphertexts together with a *sum* query calculating the sum of this column, and outputs the corresponding plaintexts. Recall that, Algorithm 1 requires the query to touch α columns to reach a successful rate of $(\sum_{i=1}^{+\infty} \frac{1}{i^\alpha})^{-1}$. The successful rate for Algorithm 2 is $(\sum_{i=1}^{+\infty} \frac{1}{i^n})^{-1}$, where n is the number of rows; and typically n is at least in the order of millions.

[1]This strategy can be applied to all attacks we present in this paper.

Therefore, via this attack, SP can always recover the plaintexts of a column with $\sim 100\%$ probability.

Algorithm 2: Co-prime attack against *sum* (for 1 column)

Input: a column of encrypted items $E(k_1, v_1)$, ..., $E(k_n, v_n)$; a *sum* query Q for this column

Output: $v_1, ..., v_n$

1 **for** $i = 1 \rightarrow n$ **do** update to the same key
2 $\quad m^{-1}v_i \leftarrow \mathsf{KeyUpdate}(E(k_i, v_i), Q)$ with the assistance of DO
3 **end**
4 **for** $i = 2 \rightarrow n$ **do** compute the ratio between v_1 and v_i
5 $\quad$ **for** $\gamma_1^{(i)} = 1 \rightarrow M$ **do**
6 $\quad\quad$ **for** $\gamma_i = 1 \rightarrow M$ **do**
7 $\quad\quad\quad$ **if** $(\gamma_i m^{-1} v_1 \mod N) = (\gamma_1^{(i)} m^{-1} v_i \mod N)$ **then**
8 $\quad\quad\quad\quad$ **go to** line 4
9 $\quad\quad\quad$ **end**
10 $\quad\quad$ **end**
11 $\quad$ **end**
12 **end**
13 $v_1 \leftarrow \mathsf{SmallestCommonMultiple}(\gamma_1^{(2)}, ..., \gamma_1^{(\alpha)})$
14 **for** $i = 2 \rightarrow \alpha$ **do**
15 $\quad v_i := v_1 \cdot \frac{\gamma_i}{\gamma_1^{(i)}}$
16 **end**
17 **return:** $v_1, ..., v_n$

Even worse, for other columns that have not been touched by a *sum* query, SP can still recover them with $\sim 100\%$ probability, as long as they are"connected" with a column touched by a *sum* query. More specifically, suppose a set of columns are touched by a set of connected queries (cf. Definition 1), based on Observation 5, these columns can be converted to be under the same column key. If a *sum* query is issued for any of these columns, Algorithm 2 can be applied to all these columns. Moreover, the co-prime attack against sum applies to *equi-join* and *group-by* as well.

However, co-prime attack against *sum* may fail to attack *group-by* and *equi-join* if items in the operated columns are so large (e.g., the items are in string type) that we cannot get ratios between them by trying each pair in $\mathbb{Z}_M^2$. However, since ciphertexts are converted to $m^{-1}v_1, ..., m^{-1}v_n$, the same plaintexts have the same ciphertexts, which reveals plaintexts frequency. Therefore, *group-by* and *equi-join* in SDB can be attacked by frequency analysis [19]. This attack is beyond the scope of this paper.

4.4 Co-prime attack against comparison

Algorithm 3 describes the co-prime attack against the *comparison* operation in SDB. In this attack, if an encrypted item has been compared for α times (no matter compared with the encrypted item in the same row, or with a constant), SP can recover it with a probability of nearly $(\sum_{i=1}^{+\infty} \frac{1}{i^\alpha})^{-1}$.

Algorithm 3: Co-prime attack against *comparison* (for 1 row)

Input: $E(k_0, v)$; $\{E(k_1, u_1), ..., E(k_\alpha, u_\alpha)\}$; α queries $\{Q_1, ..., Q_\alpha\}$, each with an operation comparing v and other values

Output: v

1 **for** $i = 1 \rightarrow \alpha$ **do** update to a given key
2 $\quad E(k_i', v) \leftarrow \mathit{KeyUpdate}(E(k_0, v), Q_i)$ with the assistance of DO
3 **end**
4 **for** $i = 1 \rightarrow \alpha$ **do** compute the product of v and u_i
5 $\quad E(k_i'', u_i v) \leftarrow E(k_i', v) \times E(k_i, u_i)$
6 $\quad u_i v \leftarrow \mathit{KeyUpdate}(E(k_i'', v_i u), Q_i)$ with the assistance of DO
7 **end**
8 $v \leftarrow \mathsf{GreatestCommondivisor}(u_1 v, ..., u_\alpha v)$
9 **return:** v

Recall that to compare v with v_i, SP executes

$$u_i(v - v_i) \leftarrow cmp(E(k_0, v), E(k_i', v_i)),$$

and compares $u_i(v - v_i)$ with $\frac{N}{2}$. According to Observation 4, SP not only gets $u_i(v - v_i)$, but also gets $u_i v$ (line 4 -6). With α queries $\{Q_1, ..., Q_\alpha\}$ for v, SP gets α products $\{u_1 v, ..., u_\alpha v\}$. Notice that $\{u_1, ..., u_\alpha\}$ are random numbers, hence w.h.p. they are co-prime when α is large. Even for $\alpha = 4$, the co-prime probability is still nearly 92.4%. Then, SP can recover v by computing the greatest common divisor of $\{u_1, ..., u_\alpha\}$ (line 8):

$$v = \mathsf{GreatestCommonDivisor}(u_1 v, ..., u_\alpha v)$$

The situation becomes even worse when we consider connected queries. Recall that the columns touched by a set of connected queries can be converted to be under any column key of these columns (c.f. Observation 5). Then, these α queries are no longer required to be issued for the same column; they can be issued for different columns as long as these columns are "connected". Furthermore, the plaintexts of *all* these columns will be recovered.

Recall that the connected queries are defined for queries involving *addition* operations. We remark that if a *comparison* query is issued for two columns (then it involves *addition* between these two columns), it can also be connected with other queries.

4.5 Summary

We have presented three attacks against five different operations in SDB. Now, we compare them and give a summary.

In the attacks against *addition*, we assume data items are under uniform distribution in plaintext space. With this assumption, we only need four columns to be added together to recover plaintexts with at least 90% probability.

In the attacks against *sum* (also *equi-join* and *group-by*), we only need to assume there are enough rows in a database so that elements in the same column are co-prime with nearly 100% probability, which always happens in real-world. With this weak assumption, we can recover all plaintexts in a column in nearly 100% probability. Besides, with more columns whose ciphertexts have been converted to be under the same column key, we can recover more columns.

In the attacks against *comparison*, we do not need any assumption about the plaintexts. We only need four comparison operations on any column of the columns whose ciphertexts can be converted to be under the same column key.

5 EXPERIMENTS

In this section, we evaluate our co-prime attack by conducting the following benchmarks:

- **Credit.** This is the "default of credit card clients" dataset taken from the UCI machine learning repository[2]. It consists 30000 rows and 25 columns, containing banking information of credit card clients in Taiwan from April to September. It did not specify queries, so we generate queries with *addition*, *comparison* and *sum* operations by ourselves.
- **TPC-C.** This is an on-line transaction processing benchmark simulating a complete environment for terminal operators to execute transactions against a database[3]. We pick 4 columns from its database: C_BALANCE, OL_O_ID, S_QUANTITY and OL_AMOUNT. In the query stream of TPC-C, there are update queries with *addition* operation querying C_ BALANCE; there are queries with *comparison* operation querying OL_O_ID and S_QUANTITY; and there are queries with *sum* operation querying OL_AMOUNT. CryptDB [22] uses this benchmark to evaluate its performance.
- **TPC-H.** This is a decision support benchmark measuring multiple aspects of the capability of a database system to process queries[4]. We pick four columns from its database: L_QUANTITY, L_DISCOUNT, PS_AVAILQTY, and L_ EXTENDEDPRICE. In the query stream of TPC-H, there are queries with *comparison* operation querying PS_AVAILQTY and L_QUANTITY; and there are queries with *sum* operation querying L_QUANTITY, L_DISCOUNT and L_ EXTENDEDPRICE. SDB [26] uses this benchmark to evaluate its performance.

The target columns and query operations in TPC benchmarks are summarized in Table 3. Besides, we give some necessary statistical information about our target columns in Table 4.

Table 3: Columns and operations in TPC benchmarks.

Attributes	Benchmark	Operation
C_BALANCE	TPC-C	*add*
OL_O_ID	TPC-C	> (<)
S_QUANTITY	TPC-C	> (<)
OL_AMOUNT	TPC-C	*sum*
PS_AVAILQTY	TPC-H	> (<)
L_QUANTITY	TPC-H	> (<) & *sum*
L_DISCOUNT	TPC-H	*sum*
L_EXTENDEDPRICE	TPC-H	*sum*

[2]https://archive.ics.uci.edu/ml/datasets.php
[3]http://www.tpc.org/tpcc/
[4]http://www.tpc.org/tpch/

Table 4: Statistical information about the datasets we use. 0 (%) are calculated with original datasets. Other information are all calculated with datasets removed 0s.

Benchmark	0 (%)	Neg (%)	Min	Max	Ave
Credit	10.1	3.2	-10	136468	4948.7
TPC-C	2.4	0	0	104900	11487.6
TPC-H	15.3	1.3	-339603	1684259	29657.2

5.1 Experimental settings

We encrypt the databases using SDB and perform co-prime attacks against their queries. To be consistent with [26], we set ρ_1 and ρ_2 to be 512 bits. Besides, to show that our attacks work under different parameter settings, we conduct experiments with larger security parameters and give results in Figure 4.

As 0s can be easily attacked (cf. Observation 1), we remove all rows that contain 0s. Moreover, larger databases (more rows and more columns) result in better attacking performance, for the following reasons:

- For the attacks against *sum/equi-join/group-by*, more rows result in higher co-prime probability, and hence higher recovery rate.
- For the attack against *addition*, more columns being added result in higher co-prime probability, and hence higher recovery rate; but independent of the number of rows.
- For the attack against *comparison*, more times a column being compared result in higher co-prime probability, and hence higher recovery rate.

To this end, we only consider small databases: we randomly pick 1000 rows and at most 12 columns from each dataset to conduct experiments. Our experimental results in Table 6 and Figure 6 validate the above statement. In Table 6, the recovery rate is at most 85.7% with 4 columns being added and it is at least 95% with 10 columns being added. In Figure 6, the recovery rate is about 92% with 4 range queries and it is almost 100% with 10 range queries.

All experiments were conducted on a Mac laptop with Intel Core i5 processor and 8GB Memory running macOS High Sierra (v10.13.6). In all of our experiments, we repeat the process 10 times.

5.2 Attacking time

We first measure the time usage of the attack against *addition* with different plaintext spaces and parameter settings (with randomly generated data). The results are shown in Figure 4: our attack against *addition* works with different plaintexts smaller than 2^{20} and can succeed under different parameter settings. Since the attack against *comparison* is influenced little by larger parameter settings (as it only iterates α times) and attack against *sum/equi-join/group-by* is similar to attack against addition, we do not repeat these experiments here.

Attacking a plaintext space of 2^{20} requires 30 minutes on our laptop. This time usage will increase linearly as the plaintext space grows (e.g., attacking a plaintext space of 2^{24} requires 8 hours), thereby attacking a larger plaintext space requires more dedicated

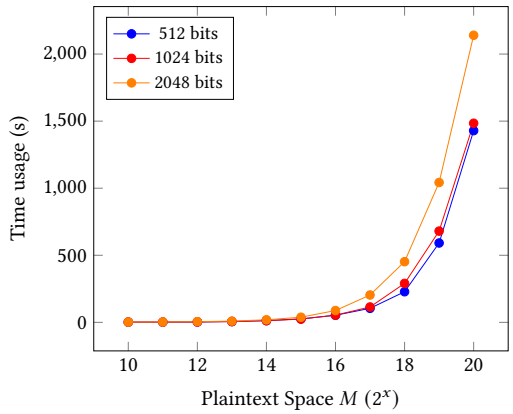


Figure 4: Time usage of co-prime attack against addition with different plaintext spaces and different parameter settings.

machines. However, we argue that 2^{24} is large enough for the common databases. To demonstrate this, we survey datasets from different fields including business, economics, education and other fields in Kaggle[5]. In each field, we pick the most popular 240 datasets and calculate the proportion of columns whose items are no more than 2^{20}, 2^{22} and 2^{24} respectively (cf. Table 5).

Table 5: The proportion of columns whose items are no more than 2^{20}, 2^{22} and 2^{24} respectively.

Field	$< 2^{20}$(%)	$< 2^{22}$(%)	$< 2^{24}$(%)
Business	94.9	95.8	96.9
Economics	83.0	85.3	86.7
Education	91.3	94.8	98.3
Health	92.0	94.8	97.5
Arts and Entertainment	91.1	93.5	95.2

5.3 Against addition

In the "credit" database, there are 12 columns that support *addition*. To evaluate the performance of our co-prime attack against *addition*, we generate queries such as:

$$SELECT\ c = \sum_{i=1}^{k} c_i.$$

We randomly pick $k = 2, ..., 12$ columns from the database and execute the above query.

For example, if we pick 5 columns from the database, the *addition* query would be performed among these 5 columns; we repeat this experiment ten times, and each time we re-select another 5 columns (if we pick 12 columns, then we have no other option but add these 12 columns once). We report max, min, avg and std of the proportion of recovered plaintexts in Table 6. Notice that the data values are not randomly distributed as we assumed in Section 4.2. Therefore, the proportion of recovered plaintexts is lower than the theoretical

[5]https://www.kaggle.com/

Table 6: Co-prime attack against *addition* in the "Credit" benchmark. Col indicates how many columns have been added together. Min, Max, Average and Stddev separately indicate the minimum, maximum, average and standard deviation of the proportion of recovered plaintexts in our experiments. Cost indicates the time usage of the attack.

# Col	Min(%)	Max(%)	Average(%)	Stddev(%)	Cost(s)
2	34.0	58.3	47.2	8.5	111.6
3	60.4	79.5	67.1	6.5	229.8
4	57.4	85.7	76.8	8.6	291.9
5	81.8	89.4	84.0	2.4	264.9
6	77.2	94.0	88.5	5.0	478.5
7	90.5	94.9	92.8	1.5	645.3
8	90.9	96.2	94.1	1.6	792.0
9	93.0	96.3	94.8	0.9	781.2
10	95.0	97.2	96.2	0.7	999.9
11	96.2	97.3	97.0	0.4	1090.0
12	97.3	97.3	97.3	\	1229.1

probability. However, there are still 76.8% plaintexts being recovered when the number of columns is 4; and it reaches 97.3% when the number of columns is 12.

The size of plaintext space in "credit" is $[0, 2^{20}]$ (most plaintexts are no more than 2^{20}; only 5 elements are over 2^{20}), hence we can try each pair in $\mathbb{Z}^k_{2^{20}}$ to find the ratio among k items, which takes 111.6-1229.1 seconds (cf. Table 6). However, the values in the "credit" benchmark are typically small so that most time we do not need to go over the whole plaintext space.

We also evaluate the performance of our co-prime attack against *addition* via the TPC-C benchmark. The *addition* operations of TPC-C are included in the update queries:

$$UPDATE\ customer\ SET\ c\_balance = c\_balance + ?.$$

where ? is a number generated in TPC-C query stream. Figure 5 shows the proportion of recovered plaintexts with different number of update queries.

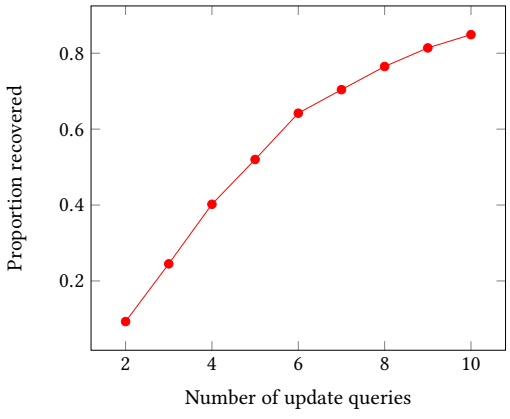


Figure 5: Proportion of recovered plaintexts of co-prime attack against addition with update queries in TPC-C.

5.4 Against sum

We evaluate our *co-prime* attack against *sum* operation using all three benchmarks. In each benchmark, we select all columns that can be queried with a *sum* operation and try to recover these columns with our co-prime attack. Here is an example of a *sum* query in TPC-C:

SELECT SUM(ol_amount) FROM order_line WHERE ol_o_id = ? AND ol_d_id = ? AND ol_w_id = ?

If we run our attack for all rows, we can certainly recover 100% plaintexts. So, we randomly choose $2 - n$ rows to run our attack. We run 10 times for each column in TPC-C and TPC-H, and record the minimum and maximum number that makes our attack success. For Credit, we run once for each column.

Table 7: Co-prime attack against *sum* in all benchmarks. Columns indicates how many columns that can be queried with sum. Recovered indicates how many columns we recovered. Min and Max separately indicate the minimum and the maximum number of rows to make our attack success.

Benchmark	Columns	Recovered	Min	Max
Credit	12	12	2	5
TPC-C	1	1	2	6
TPC-H	3	3	2	5

The co-prime attacks against *equi-join* and *group-by* are identical as that against *sum*. They can also recover 100% plaintexts as long as the number of rows is enough. So, we omit these experiments.

5.5 Against comparison

Recall that the success rate of our co-prime attack against comparison is independent of the data distribution; but only depends on the co-prime probability of the random masks $\{u_1, ..., u_\alpha\}$ [6] . Then, for each benchmark we only need to evaluate one column; the attack performance on other columns should be similar. In the "credit" benchmark, we select one column and generate 2-10 range queries for this column. In the TPC-C and TPC-H benchmarks, we select one column from each benchmark, and pick the first 2-10 range queries regarding this column from the query stream. Here is an example of a range query in TPC-C:

SELECT count(*) FROM stock WHERE s_w_id = ? AND s_i_id = ? AND s_quantity < ?

Figure 6 shows the proportion of recovered plaintexts of co-prime attack against comparison with different number of range queries. In all these three benchmarks, more than 90% plaintexts are recovered with 4 range queries to the same column; and almost 100% plaintexts are recovered with 10 range queries to the same column. So the range queries in SDB is fragile under co-prime attack against comparison.

[6]To keep consistent with SDB, each of them has 80 bits.

6 COUNTERMEASURES

Recall that the attacks against *addition* and *sum/equi-join/group-by* require SP to iterate the plaintext space to find the ratio. The basic idea of our countermeasures is to enlarge the plaintext space by multiplying a random value to each data item, so that SP can no longer iterate it. Data items in the same row should be multiplied with the *same* value for two reasons: (1) additions can still be done, and (2) DO does not need to store random values for all items in the database. However, this is not enough to prevent the attack against addition, as SP can still iterate the original plaintext space to find the ratio. To this end, we also add noise to the new plaintext. The encryption scheme becomes:

Encryption. DO encrypts a data item v as:

$$c := (mg^{rx})^{-1} \cdot (av + b) \mod N,$$

where $0 \ll b \ll a$ and $M \ll av + b \ll N$. The column key is $\langle m, x \rangle$ and the row key is $\langle r, a \rangle$. DO does not store b.

Multiplication: Given two ciphertexts $E(k_1, av_1+b_1)$ and $E(k_2, av_2+b_2)$ in the same row, SP computes:

$$E(k_3, a^2v_1v_2+(av_1b_2+av_2b_1+b_1b_2)) \leftarrow E(k_1, av_1+b_1)\times E(k_2, av_2+b_2).$$

Notice that a must be large enough so that $av_1b_2+av_2b_1+b_1b_2 \ll a^2$, and $(av_1b_2 + av_2b_1 + b_1b_2)$ is considered as the new "b".

KeyUpdate: The *KeyUpdate* function is the same as before.

Addition. Given two ciphertexts $E(k_1, av_1 + b_1)$ and $E(k_2, av_2 + b_2)$ in the same row, DO first convert them to be under the same key k_3 then adds them together:

$$E(k_3, a(v_1 + v_2) + (b_1 + b_2)) \leftarrow E(k_1, av_1 + b_1) + E(k_2, av_2 + b_2).$$

Decryption. Given a ciphertext $c = (a^t v + b) \cdot (mg^{rx})^{-1}$, DO decrypts it as follows:

$$m \leftarrow \left\lfloor \frac{c \cdot mg^{rx} \mod N}{a^t} \right\rfloor$$

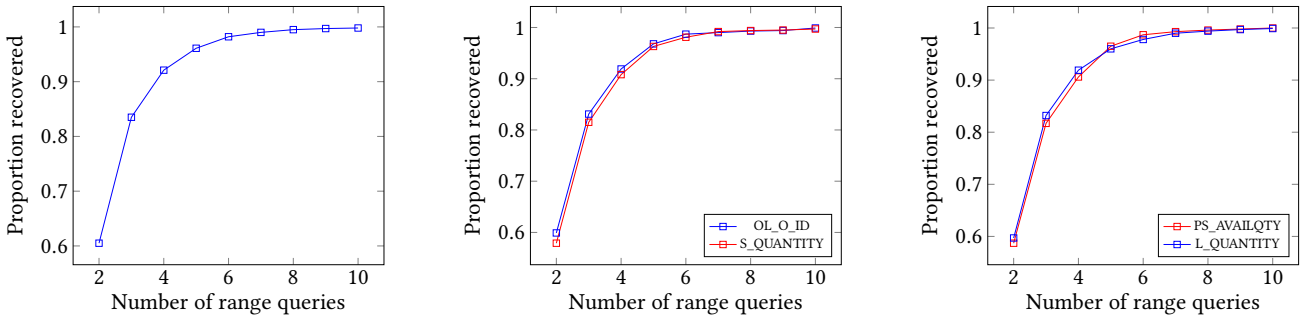
where t is the number of times a ciphertext has been multiplied. Decryption can succeed because b is much smaller than a^t so that it will be canceled by the "rounding" operation.

Notice that this new encryption scheme can only support a limited number of additions and multiplications: *addition* might make $0 \ll b \ll a$ no longer hold and *multiplication* might make $M \ll av + b \ll N$ no longer hold. Furthermore, this scheme cannot securely support comparisons, as the attack against *comparison* does not require SP to iterate the plaintext space (cf. Algorithm 3). We leave it as a future work to refine this encryption scheme.

7 DISCUSSION

7.1 Lesson learned

SDB is in fact a multiplicatively-homomorphic encryption scheme of the form: $(mg^{rx})^{-1} \cdot v$, where $\langle \langle m, x \rangle, r \rangle$ can be considered as the randomness, because for any two different ciphertexts, they either have different $\langle m, x \rangle$ or different r. To add some ciphertexts, one has to make them with the same $\langle \langle m, x \rangle, r \rangle$, otherwise, addition cannot be done. Then, SDB can be considered as an encrypted database, where the ciphertexts are encrypted under the same key. We prove the following theorem for such kind of encrypted databases:



Bank database (selected one column) **TPC-C (two columns)** **TPC-H (two columns)**

Figure 6: Proportion of recovered plaintexts of co-prime attack against comparison with different number of range queries.

Theorem 2. *In an encrypted database where the ciphertexts are encrypted under the same key, if a Leakage function $L(\cdot)$ can be applied to a single column of ciphertexts with sublinear communication, then it can be applied to all ciphertexts.*

Proof. We define the leakage function $L(\cdot)$ as:

$$f(m) \leftarrow L(c),$$

where c is the ciphertext, m is the plaintext, and $f(m)$ is the leaked information. Consider the following two cases:

(1) The encryption is deterministic. Then, it is clear that $L(\cdot)$ can be applied to all other ciphertexts, as they are encrypted under the same key.
(2) The encryption is non-deterministic, which means that each ciphertext has different randomness. Suppose there are n ciphertexts in a column. Let β denote the information sent from DO to SP. As the communication is sublinear, β can be customized for at most m ($m < n$) ciphertexts in this column. Recall that $L(\cdot)$ can be applied to the whole column, which means β is applicable to the rest $(n - m)$ ciphertexts as well. That means the randomness in the ciphertexts has no effect on β. Then, β can be applied to the ciphertexts in other columns.

□

In SDB, the leakage function for addition is $(k_0 v_1, ..., k_0 v_\alpha)$, where $k_0 = (mg^{rx})^{-1}$. SP has to iterate the plaintext space to find ratio and recover v_i. We can prevent the co-prime attack by enlarging the plaintext space (cf. Section 6). However, the leakage function for comparison is $(u_0 v, ..., u_\alpha v)$ and SP can easily find v. In summary, we point out the following two research directions for preventing the co-prime attack:

- Encrypt different columns using different keys. Then, we need to explore how to support interoperability.
- Try to minimize the leakage function!

7.2 Attacking other schemes

We remark that our co-prime attack is *not* specific to SDB. Instead, it can also be applied to other encryption schemes having similar favors with SDB. For example, our co-prime attack can be applied to an encryption scheme [8] that has been widely used in outsourced computation [4, 10, 12, 25]. It works as follows:

- **Setup:**

$$\{(N', d), (M', r)\} \leftarrow Gen(1^\lambda)$$

(N', d) are public while (M', r) are private. N' should have many small divisors and there should be many elements less than N' that are co-prime with it. M' is a divisor of N' and $r \in \mathbb{Z}_{N'}$ is co-prime with N'.

- **Encryption:**

$$c \leftarrow Enc(v, r, d, N', M')$$

divides v into a d-tuple $(v^{(1)}, v^{(2)}, ..., v^{(d)}) \in \mathbb{Z}_{N'}^d$ such that $v = \sum_{i=1}^{d} v^{(i)} \mod M'$; the ciphertext is

$$c = (v^{(1)} \cdot r, v^{(2)} \cdot r^2, ..., v^{(d)} \cdot r^d) \mod N'$$

- **Decryption:**

$$v \leftarrow Dec(c, r, N', M')$$

the plaintext is calculated as:

$$(v^{(1)}, ..., v^{(d)}) = (c^{(1)} r^{-1}, ..., c^{(d)} r^{-d}) \mod N'$$

$$v = \sum_{i=1}^{d} v^{(i)} \mod M'$$

- **Addition:**

$$E(v_1 + v_2) \leftarrow Add(c_1, c_2)$$

computes $E(v_1 + v_2)^{(k)} = c_1^{(k)} + c_2^{(k)} \mod N'$

- **Multiplication:**

$$E(v_1 \cdot v_2) \leftarrow Mult(c_1, c_2)$$

computes $E(v_1 \cdot v_2)^{(k)} = \sum_{i+j=k+1} c_1^{(i)} c_2^{(j)} \mod N'$

Due to co-prime probability (cf. Section 4.1), this scheme can be attacked under known-plaintext attack, which was introduced in [7, 24]. Suppose there are $d + 2$ known pairs $(v_i, E(v_i))$, the adversary recovers (M', r) by constructing the following equations:

$$\begin{bmatrix} v_1 & c_1^{(1)} & \cdots & c_1^{(d)} \\ v_2 & c_2^{(1)} & \cdots & c_2^{(d)} \\ \vdots & \vdots & & \vdots \\ v_{d+1} & c_{d+1}^{(1)} & \cdots & c_{d+1}^{(d)} \end{bmatrix} \begin{bmatrix} -1 \\ r^{-1} \\ \vdots \\ r^{-d} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \mod M' \quad (1)$$

Denote this coefficient matrix as A_1. As the equation has a nontrivial solution $(-1, r^{-1}, ..., r^{-d}) \mod M' \in Z_{M'}^{d+1}$, we have $det(A_1) \equiv 0 \mod M'$, which means $det(A_1)$ is a multiple of M'. With $d+2$ known pairs, the adversary can construct $\binom{d+2}{d+1} = d+2$ determinants: $(A_1, A_2, ..., A_{d+2})$. Suppose there are k determinants $(A_1, .., A_k)(2 \leq k \leq d+2)$ satisfy $det(A_i) \neq 0$. As $(\frac{det(A_1)}{M'}, ..., \frac{det(A_k)}{M'})$ are co-prime w.h.p, M' can be recovered:

$$M' \leftarrow gcd(det(A_1), det(A_2), ..., det(A_k))$$

With M', the adversary can get r' that satisfies $r \equiv r' \mod M'$ by solving Equation 1. Although the original scheme decrypts ciphertexts with r, r' can be used for decryption as well.

8 RELATED WORK

Most encrypted databases (e.g., CryptDB [22], MONOMI [23]) leverage property-preserving encryption (PPE) to support comparisons. Deterministic encryption (DET) encrypts the same plaintexts to the same ciphertexts, so that equal operation, join operation and group-by operation can be done. Order preserving encryption (OPE) [3, 6] preserves the order information of plaintexts, so that range queries and comparison operations can be processed. PPE-based encrypted databases like CryptDB are vulnerable to many attacks since sensitive information are leaked from ciphertexts: OPE leaks order information of plaintexts and DET leaks which plaintexts are equal. Many attacks have been proposed, we compare some of them with our *Co-prime* attack.

Leakage from ciphertexts. PPE inherently leaks information about plaintext such as frequency and order, which has been exploited extensively by various of attacks. Naveed [19] attack DET- and OPE-encrypted databases based on frequency analysis and sorting. They estimate plaintext frequency and order in advance so that they can compare frequency and order to find a map from ciphertexts to plaintexts. Cash et al. [11] adapt the attack procedure as a min-weight non-crossing bipartite matching, attacking a widely used OPE scheme [6]. Compared with [19], the non-crossing attack performs better when plaintext data are drawn from large domains. Bindschaedler et al. [5] present a new inference technique called *multinomial attack* against property-revealing encryption(PRE) schemes. Considering correlation between columns, they construct the attack based on Bayesian inference problem in multi dimensions. Besides, with the correlation and plaintexts gotten, they can infer columns protected by semantically secure encryption or redaction with machine learning and record linkage methods.

Leakage from queries. When DO issues a query, what SP returns may leak information about plaintexts. Such attack usually need plenty of queries to recover plaintexts and most of them assume queries are under known distribution. Kellaris et al. [13] develop a generic *reconstruction attack* on any encrypted database supporting range queries where either access pattern (which elements are returned) or communication volume (how many elements are returned) is leaked. However, it assumes the query distribution is known and it requires at least $O(M^2 \log M)$ (M is the number of distinct plaintext values) range queries. [15] improves $O(M^2 \log M)$ to $M \log M + O(M)$ and presents an approximate reconstruction attack which only needs the access pattern leakage of $O(M)$ queries. It can recover plaintext values in a dense dataset within a constant ratio of error.

Kornaropoulos et al. [14] present a reconstruction attack that succeeds without any knowledge about the query or data distribution. Based on the search-pattern leakage and a technique named support size estimation, they use range queries and k-nearest-neighbor(k-*NN*) queries to reconstruct plaintext values under a variety of skewed query distributions. However, it still requires queries to be under some fixed distributions. For example, queries should touch all plaintext values, if all queries only touch plaintexts from $[0, M/2]$ (only plaintexts in this interval are returned), then plaintexts in $[M/2, M]$ cannot be recovered.

Comparison. We compare these attacks with our co-prime attack in the context of attacking SDB. Recall that the *sum*, *equi-join* and *group-by* operations in SDB leak plaintext frequency in the same column. So attacks based on plaintext frequency [5, 11, 19] can be applied to SDB, However, auxiliary information about the plaintext frequency is required. Attacks based on range queries and *k-NN* queries [13–15] can be applied to SDB, because the *comparison* operation in SDB leaks access patterns and volume information. However, at least $M \log(M) + O(M)$ queries are required for full reconstruction and $O(M)$ queries are required for approximate reconstruction in [15].

Compared with these attacks, our co-prime attack can attack more operations in SDB and needs no more than 10 queries to recover all plaintexts. Besides, our assumption is much weaker: we only assume plaintexts are under randomly uniform distribution (our attack still works when this assumption is not satisfied).

9 CONCLUSION

In this paper, we revisit the security of SDB and make four observations, which incur serious information leakage but was not mentioned in their paper. We exploit these observations and propose a ciphertext-only attack named co-prime attack. We show how to use it to attack the *addition, sum, comparison, join and group-by* operations in SDB. We evaluate our attack in three real-world benchmarks. For columns that support *addition* and *comparison*, we recover 84.9% – 99.9% plaintexts. For columns that support *sum*, *equi-join* and *group-by*, we recover 100% plaintexts. Nevertheless, we still think the data interoperability is an important issue, and we admit that there are a bunch of smart designs in SDB. In future work, we will attempt to fix the security flaws of SDB and propose an encrypted database that supports data interoperability.

ACKNOWLEDGMENTS

The work was supported in part by Zhejiang Key R&D Plans (Grant No. 2021C01116, 2019C03133), National Natural Science Foundation of China (Grant No. 62002319, U20A20222) as well as a grant from China Zheshang Bank.

REFERENCES

[1] [n.d.]. GCreep: Google engineer stalked teens, spied on chats. Gawker. http://gawker.com/5637234/. Accessed in December 2020.

[2] [n.d.]. National Vulnerability Database. CVE statistics. https://nvd.nist.gov/vuln/search. Accessed in December 2020.

[3] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. 2004. Order Preserving Encryption for Numeric Data. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data* (Paris, France) *(SIGMOD '04)*. Association for Computing Machinery, New York, NY, USA, 563–574. https://doi.org/10.1145/1007568.1007632

[4] A. Alabdulatif, Heshan Kumarage, I. Khalil, and X. Yi. 2017. Privacy-preserving anomaly detection in cloud with lightweight homomorphic encryption. *J. Comput. Syst. Sci.* 90 (2017), 28–45.

[5] Vincent Bindschaedler, Paul Grubbs, David Cash, Thomas Ristenpart, and Vitaly Shmatikov. 2018. The Tao of Inference in Privacy-Protected Databases. *Proc. VLDB Endow.* 11, 11 (July 2018), 1715–1728. https://doi.org/10.14778/3236187.3236217

[6] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. 2009. Order-Preserving Symmetric Encryption. In *Advances in Cryptology - EUROCRYPT 2009*, Antoine Joux (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 224–241.

[7] Jung Cheon and Hyun Nam. 2003. A Cryptanalysis of the Original Domingo-Ferrer's Algebraic Privacy Homomophism. *IACR Cryptology ePrint Archive* 2003 (01 2003), 221.

[8] Josep Domingo-Ferrer. 2002. A Provably Secure Additive and Multiplicative Privacy Homomorphism*. In *Information Security*, Agnes Hui Chan and Virgil Gligor (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 471–483.

[9] Ariel J. Feldman, William P. Zeller, Michael J. Freedman, and Edward W. Felten. 2010. SPORC: Group Collaboration using Untrusted Cloud Resources. In *9th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2010, October 4-6, 2010, Vancouver, BC, Canada, Proceedings*, Remzi H. Arpaci-Dusseau and Brad Chen (Eds.). USENIX Association, 337–350. http://www.usenix.org/events/osdi10/tech/full_papers/Feldman.pdf

[10] J. Girao, D. Westhoff, and M. Schneider. 2005. CDA: concealed data aggregation for reverse multicast traffic in wireless sensor networks. In *IEEE International Conference on Communications, 2005. ICC 2005. 2005*, Vol. 5. 3044–3049 Vol. 5. https://doi.org/10.1109/ICC.2005.1494953

[11] P. Grubbs, K. Sekniqi, V. Bindschaedler, M. Naveed, and T. Ristenpart. 2017. Leakage-Abuse Attacks against Order-Revealing Encryption. In *2017 IEEE Symposium on Security and Privacy (SP)*. 655–672. https://doi.org/10.1109/SP.2017.44

[12] H. Hu, J. Xu, C. Ren, and B. Choi. 2011. Processing private queries over untrusted data cloud through privacy homomorphism. In *2011 IEEE 27th International Conference on Data Engineering*. 601–612. https://doi.org/10.1109/ICDE.2011.5767862

[13] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. 2016. Generic Attacks on Secure Outsourced Databases. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 1329–1340. https://doi.org/10.1145/2976749.2978386

[14] E. M. Kornaropoulos, C. Papamanthou, and R. Tamassia. 2020. The State of the Uniform: Attacks on Encrypted Databases Beyond the Uniform Query Distribution. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 1223–1240. https://doi.org/10.1109/SP40000.2020.00029

[15] M. Lacharité, B. Minaud, and K. G. Paterson. 2018. Improved Reconstruction Attacks on Encrypted Data Using Range Query Leakage. In *2018 IEEE Symposium on Security and Privacy (SP)*. 297–314. https://doi.org/10.1109/SP.2018.00002

[16] Derrick Norman Lehmer. 1900. Asymptotic Evaluation of Certain Totient Sums. *American Journal of Mathematics* 22, 4 (oct 1900), 293. https://doi.org/10.2307/2369728

[17] Jinyuan Li, Maxwell Krohn, David Mazières, and Dennis Shasha. 2004. Secure Untrusted Data Repository (SUNDR). In *Proceedings of the 6th Conference on Symposium on Operating Systems Design amp; Implementation - Volume 6* (San Francisco, CA) *(OSDI'04)*. USENIX Association, USA, 9.

[18] Prince Mahajan, Srinath Setty, Sangmin Lee, Allen Clement, Lorenzo Alvisi, Mike Dahlin, and Michael Walfish. 2011. Depot: Cloud Storage with Minimal Trust. *ACM Trans. Comput. Syst.* 29, 4, Article 12 (Dec. 2011), 38 pages. https://doi.org/10.1145/2063509.2063512

[19] Muhammad Naveed, Seny Kamara, and Charles V. Wright. 2015. Inference Attacks on Property-Preserving Encrypted Databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (Denver, Colorado, USA) *(CCS '15)*. Association for Computing Machinery, New York, NY, USA, 644–655. https://doi.org/10.1145/2810103.2813651

[20] J.E Nymann. 1972. On the probability that k positive integers are relatively prime. *Journal of Number Theory* 4, 5 (1972), 469–473. https://doi.org/10.1016/0022-314X(72)90038-8

[21] Pascal Paillier. 1999. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Advances in Cryptology — EUROCRYPT '99*, Jacques Stern (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 223–238.

[22] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2011. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (Cascais, Portugal) *(SOSP '11)*. Association for Computing Machinery, New York, NY, USA, 85–100. https://doi.org/10.1145/2043556.2043566

[23] Stephen Tu, M. Frans Kaashoek, Samuel Madden, and Nickolai Zeldovich. 2013. Processing Analytical Queries over Encrypted Data. *Proc. VLDB Endow.* 6, 5 (March 2013), 289–300. https://doi.org/10.14778/2535573.2488336

[24] David Wagner. 2003. Cryptanalysis of an Algebraic Privacy Homomorphism. In *Information Security*, Colin Boyd and Wenbo Mao (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 234–239.

[25] D. Westhoff, J. Girao, and M. Acharya. 2006. Concealed Data Aggregation for Reverse Multicast Traffic in Sensor Networks: Encryption, Key Distribution, and Routing Adaptation. *IEEE Transactions on Mobile Computing* 5, 10 (2006), 1417–1431. https://doi.org/10.1109/TMC.2006.144

[26] Wai Kit Wong, Ben Kao, David Wai Lok Cheung, Rongbin Li, and Siu Ming Yiu. 2014. Secure Query Processing with Data Interoperability in a Cloud Database Environment. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (Snowbird, Utah, USA) *(SIGMOD '14)*. Association for Computing Machinery, New York, NY, USA, 1395–1406. https://doi.org/10.1145/2588555.2588572