# SAGMA: Secure Aggregation Grouped by Multiple Attributes

Timon Hackenjos*
FZI Research Center for Information Technology
hackenjos@fzi.de

Florian Hahn*
University of Twente
f.hahn@utwente.nl

Florian Kerschbaum
University of Waterloo
fkerschb@uwaterloo.ca

## ABSTRACT

Encryption can protect data in outsourced databases – in the cloud – while still enabling query processing over the encrypted data. However, the processing leaks information about the data specific to the type of query, e.g., aggregation queries. Aggregation over user-defined groups using SQL's `GROUP BY` clause is extensively used in data analytics, e.g., to calculate the total number of visitors each month or the average salary in each department. The information leaked, e.g., the access pattern to a group, may reveal the group's frequency enabling simple, yet detrimental leakage-abuse attacks.

In this work we present *SAGMA* – an encryption scheme for performing secure aggregation grouped by multiple attributes. The querier can choose any combination of one or multiple attributes in the `GROUP BY` clause among the set of all grouping attributes. The encryption scheme only stores semantically secure ciphertexts at the cloud and query processing hides the access pattern, i.e., the frequency of each group. We implemented our scheme and our evaluation results underpin its practical feasibility.

## CCS CONCEPTS

• **Security and privacy → Management and querying of encrypted data**; *Privacy-preserving protocols.*

## KEYWORDS

Encrypted Databases; Secure Data Aggregation

*Parts of this work have been done while at SAP Security Research.


*SIGMOD '20, June 14–19, 2020, Portland, OR, USA*

ACM ISBN 978-1-4503-6735-6/20/06…$15.00
https://doi.org/10.1145/3318464.3380569

**ACM Reference Format:**
Timon Hackenjos, Florian Hahn, and Florian Kerschbaum. 2020. SAGMA: Secure Aggregation Grouped by Multiple Attributes. In *2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3318464.3380569

## 1 INTRODUCTION

Outsourcing databases and analytics to third-party cloud providers is commonplace in enterprise computing. Customers of cloud computing profit from a more flexible pay-per-use cost model than available for in-house computing centers. Outsourcing, however, implies novel security threats, i.e., the cloud service provider must be trusted.

Encryption at the client side can be used to mitigate those security threats. Standard encryption schemes, such as AES-GCM or RSA-OAEP, render database operations at the cloud provider impossible. Fully homomorphic encryption (FHE) enables the execution of arbitrary queries on encrypted data, but is currently still too inefficient for most practical use [1, 15]. Secure multi-party computation (MPC), e.g., SMCQL [3], requires multiple, mutually distrustful cloud providers increasing cost and administrative burden. Cloud providers can use trusted hardware modules, such as Intel's SGX [8, 27], but this only shifts the trust anchor from the cloud provider to the hardware manufacturer. Recent bugs in the Intel Management Engine [10] and bugs based on speculative execution [5] create doubts in the security those hardware modules provide. An active line of research addresses these shortcomings by developing encryption schemes that enable query processing over encrypted data, e.g., [13, 17, 25, 26].

Recently, Kamara and Moataz [22] introduced the first system supporting a large class of SQL queries without using property-preserving encryption. However, their proposal lacks an efficient solution for data aggregation on the server side but they propose data aggregation on the client side after data filtering and decryption.

In this work, we bridge this gap and consider aggregation queries grouped by multiple attributes over encrypted data combined with filtering. Aggregation queries are the most common form of query in data analytics. For example, Piwik, a popular web analytics tool, which – among others – we use

to evaluate our encryption scheme, determines the number of visitors of a site by country, browser, referrer, time and many other attributes.

Designing a secure aggregation scheme is challenging, since the tuples composing a group are aggregated into one result, i.e., the access pattern of an aggregation query reveals the frequency of each group. Currently, there exist two approaches in encryption schemes for secure aggregation. First, *revealing the access pattern*: Deterministic encryption, e.g., used in CryptDB [26] allows grouping on ciphertexts in combination with partial homomorphic encryption for data aggregation. This approach allows grouping by any combination of attributes, but leaks the frequency of each grouping attribute value. Simple, yet efficient leakage-abuse attacks exploit this frequency information to reconstruct plaintext values with high accuracy as recent work demonstrates [16, 23]. Second, *pre-computing the groups into an index*: Seabed [25] thwarts frequency analysis attacks by flattening the histogram of attribute values. It uses multiple columns per attribute and inserts dummy tuples. However, all group values and possible combinations of grouping attributes in queries must be known at encryption time and pre-computed to be outsourced; thus increasing server storage requirements exponentially. Further, filtering clauses (SQL's `WHERE`) must be mostly handled at the client.

In this work, we present *SAGMA* – secure aggregation grouped by multiple attributes secure against a persistent, honest-but-curious adversary controlling the service provider. It reveals only semantically secure ciphertexts to the cloud and hides the frequency of attribute values during query processing. The querier can choose any combination of at most $t$ grouping attributes where $t$ is a threshold chosen at setup time. At a high level, we use somewhat homomorphic encryption to compute a ciphertext-packed aggregation of a linear combination of attribute values. Grouping attribute values are mapped to buckets balancing the storage and computational requirements. This approach supports additional filtering clauses without relying on property-preserving encryption while significantly reducing the storage requirements compared to previous solutions. We summarize the contribution of our work as follows:

- We present *SAGMA*, an encryption scheme for secure aggregation grouped by multiple attributes, that enables the querier to choose any combination of $t$ or less grouping attributes.
- We rigorously prove the security of SAGMA using a simulator of the real view of a cloud service provider using precisely defined leakage.
- We present the results of an evaluation of an implementation of SAGMA and queries from real-world systems Nextcloud, WordPress and Piwik.

## 2 PROBLEM DESCRIPTION

In the remainder of this work we assume a relational database table with the following general layout (we use the terms column and attribute interchangeably):

(1) We assume at least one column, called *value column*, to be aggregated (e.g., summed, counted, averaged),
(2) further, we assume one or multiple attributes, called *group columns*, the GroupBy clause is executed on,
(3) and finally, we assume zero or multiple auxiliary attributes, called *filtering columns*, additional filtering clauses are executed on.

One can define group and value columns as filtering columns as well. We give an example sketched in Table 1 to demonstrate this setting. Here, "Salary" is a value column, "Gender" and "Department" are group columns and "Name" and "Department" are filtering columns. A possible query formulated in SQL is given in Listing 1 yielding the corresponding result depicted in Table 2. We revise previous approaches before we outline SAGMA and briefly highlight the differences. We refer to Section 6.2 for a thorough comparison.

| ID | Salary | Gender | Name | Department |
|---|---|---|---|---|
| 1 | 1000 | male | Henry | Sales |
| 2 | 5000 | female | Jessica | Sales |
| 3 | 1500 | female | Alice | Finance |
| 4 | 3000 | male | Bob | Sales |
| 5 | 2000 | male | Paul | Facility |

**Table 1: Example table supported by our construction.**

**SELECT SUM**(Salary), Gender, Department **FROM** Example
**WHERE** Department = ''Sales'' **GROUP BY** Gender, Department;

**Listing 1: Example SQL query**

| SUM(Salary) | Gender | Department |
|---|---|---|
| 5000 | female | Sales |
| 4000 | male | Sales |

**Table 2: Result of executing SQL 1 on Table 1.**

One approach for secure aggregation in previous work reveals the access pattern. For example, CryptDB [26] supports data aggregation with arbitrary GroupBy attributes and combinations thereof on encrypted data. The evaluation of the GroupBy clauses is based on deterministic encryption of values in group columns and additively homomorphic encryption of values in value columns. Deterministic encryption $\mathrm{Enc}_{det}(\cdot)$ preserves equality: given two plaintexts $a$ and $b$ with $a = b$ it holds that $\mathrm{Enc}_{det}(a) = \mathrm{Enc}_{det}(b)$. We denote the encryption of a plaintext value $x$ using an additively homomorphic encryption scheme by $[\![x]\!]_{\oplus}$. Given two ciphertexts $[\![x]\!]_{\oplus}$ and $[\![y]\!]_{\oplus}$ one can compute the encrypted sum of the underlying plaintexts using the operation $\oplus$ on

ciphertexts: $[\![x]\!]_{\oplus} \oplus [\![y]\!]_{\oplus} = [\![x + y]\!]_{\oplus}$. In combination, the DBMS can perform GroupBy operations on encrypted data, e.g., grouping by $\text{Enc}_{det}(a)$ and subsequently perform aggregation operations for each such group using the operation $\oplus$. While additively homomorphic encryption offers semantic security, the security of deterministic encryption is questionable. Deterministic encryption leaks joint group membership for all rows enabling an attacker to reconstruct a histogram of all group values. In many cases, this histogram enables simple, yet powerful attacks as shown by Naveed et al. [23].

An alternative approach with the goal to address this security issue for aggregation is to compute (and encrypt) an index over the group columns as proposed by Papadimitriou et al. with Seabed [25]. Seabed also uses a combination of deterministic and additively homomorphic encryption. However, by splitting group attributes into multiple columns and introducing dummy elements they flatten the leaked plaintext frequencies thwarting frequency analysis. Compared to CryptDB, this approach provides better security for the grouping values; however, it restricts the choice of attribute combinations in one GroupBy statement. Further, statements containing additional filtering clauses in combination with data aggregation requires computation effort by the client linear in the filtering result size.

The naïve strategy is pre-computing the aggregation results for combinations of group column values initially during encryption time. However, there are an exponential number of such combinations of group column values requiring excessive storage space. Considering additional filtering statements, the number of potential results to be precomputed becomes impractical.

Our novel encryption scheme *SAGMA* does not unveil the access pattern for individual grouping values. At the same time, SAGMA enables queries with GroupBy clauses over multiple group columns and arbitrary number of value attributes together with support for additional filtering statements; one example for such query is given in Listing 1. In details, SAGMA employs row-wise encryption such that it can be easily combined with searchable encryption schemes, e.g. supporting filtering for specific keywords [9], ranges or substrings [11, 18] and is even compatible with complete systems [22]. Hence, it is possible to process queries by first executing the filtering operation and then using SAGMA on the result set yielded by the access pattern leakage for secure aggregation. We emphasize that this filtering is not feasible for secure aggregation schemes based on pre-built indexes or pre-computed results.

# 3 CONSTRUCTIONS

In this section, we develop the ideas for our construction for *SAGMA* step-by-step before we give a formal description of these ideas in Section 4. Our constructions use ciphertext packing initially published by Ge et al. [14] for performance improvements of additively homomorphic encryption. We divide the plaintext into several blocks enabling encryption of multiple values in one single ciphertext. One can determine these blocks either during the initial encryption step or during the actual data aggregation as elaborated in the following subsections. Applying homomorphic addition allows us to add values componentwise.

## 3.1 Initial Static Shifting

We re-use the idea of ciphertext packing, however, instead of increased performance we aim for increased security. Basically, each ciphertext consists of multiple blocks and during aggregation each block contains the current subtotal for one group attribute value. More specifically, we interpret the plaintext space $\mathcal{M}$ of an additively homomorphic encryption scheme as multiple separate *value blocks* with value domain $D_V$. For example, using secure parameters the plaintext space $\mathcal{M}$ of the additively homomorphic encryption scheme published by Pailler [24] has a size of 2048 bits, and a value domain $D_V$ has a size of 32 bits corresponding to the common integer size. We encode the value $v$ and a group attribute $g$ (of small sized group attribute domain $D$) in a transformed value $v'$ to be encrypted afterwards: The group attribute value is encoded by the index of the block containing value $v$. All remaining blocks are set to zero. Given the group attribute domain, e.g., $D = \{\text{male}, \text{female}\}$ as sketched in Table 1 and value domain $D_V = \{0, \ldots, 2^{32} - 1\}$ we can encode the tuples (1000, male), (5000, female), (1500, female), (3000, male), (2000, male) as given in the following Figure 1, where each block has bitlength 32.

| | | | |
|---|---|---|---|
| 0 | 1000 | $\Rightarrow$ | $[\![1000]\!]_{\oplus}$ |
| 5000 | 0 | $\Rightarrow$ | $[\![5000 \cdot 2^{32}]\!]_{\oplus}$ |
| 1500 | 0 | $\Rightarrow$ | $[\![1500 \cdot 2^{32}]\!]_{\oplus}$ |
| 0 | 3000 | $\Rightarrow$ | $[\![3000]\!]_{\oplus}$ |
| 0 | 2000 | $\Rightarrow$ | $[\![2000]\!]_{\oplus}$ |

**Figure 1: Example encoding for tuples consisting of value and group attributes as given in Table 1.**

From a mathematical point of view, we use a *mapping function* $f : D \rightarrow \{0, \ldots, |D|\}$ mapping group attributes to positive integers and use $f$ to determine the *blockwise left shift* $s$ encoding the group membership of value $v \in D_V$ into one transformed value $v'$ by multiplication

$$v' = v \cdot s(g) = v \cdot |D_V|^{f(g)}.$$

This transformed value $v'$ is then encrypted using an additively homomorphic encryption scheme resulting in ciphertext $[\![v']\!]_{\oplus}$. The sum over all data in combination with

GroupBy statement is then transformed to a general aggregation over encrypted data computed by additively homomorphic encryption. The client can decrypt the result and extract the individual group totals by extracting the corresponding block. This approach increases security for individual group values as it hides their access pattern.

While this scheme inherits security of the additively homomorphic encryption scheme, the group attribute domain size is restricted: the number of distinct group attribute values must be smaller than the maximum number of value blocks fitting in the plaintext domain, i.e. $\left\lceil \frac{|\mathcal{M}|}{|D_V|} \right\rceil \geq |D|$.

This constraint can be addressed by concatenating multiple ciphertexts and executing homomorphic addition componentwise. That is, given an additively homomorphic encryption scheme with plaintext size $|\mathcal{M}|$ divided in $b$ blocks (i.e. $\left\lceil \frac{|\mathcal{M}|}{|D_V|} \right\rceil = b$), hence supporting group attribute domains up to size $b$, we can extend the size by factor $n$ by concatenating $n$ plaintext messages each encrypted separately, i.e. $m' = m_1, m_2, \ldots, m_n \in \mathcal{M}^n$. As one major drawback, however, this construction results in additional storage overhead where most parts of $m'$ contain zeros but still provides semantic security.

## 3.2 Statically Shifted Bucketization

To reduce the storage overhead, we divide the complete group attribute domain $D$ in separate buckets each with bucket size $B$. Hence, each transformed value only consists of $B$ instead of $|D|$ blocks and thus requires less storage space. Aggregation is performed in each bucket separately but discloses the bucket membership for each row. Still, values of the same bucket are indistinguishable for an adversary.

To map the group attribute value to one of the $B$ blocks we use a simple modulo operation. The value $v$ is shifted to the appropriate block by multiplying it by the shift $s(g) = |D_V|^{f(g) \bmod B}$. Here, the mapping function $f$ can be seeded with an additional secret key, preventing (and hiding) coherent group values to be mapped to the same bucket.

Note that the bucket membership of rows itself can be protected, e.g., using searchable symmetric encryption with support for Boolean search queries such as published by Cash et al. [6], for databases where the aggregation is only computed over a subset of the complete outsourced data set determined by additional filtering attributes. This supplementary protection unveils the same bucket membership only for rows matching the additional filtering clause. The bucket size $B$ is an additional parameter providing the possibility to trade security for required storage space for each encrypted value as well as computation time since the aggregation is executed componentwise. We give a more detailed security analysis on the bucket size $B$ and bucketing strategies in Section 5.

## 3.3 Dynamically Shifted Bucketization

So far, we have only addressed secure aggregation protocols for databases containing a single value attribute to be aggregated, however, we strive to generalize our construction to provide functionality for multiple different columns to be aggregated. For simplicity, we assume them to have the same value domain $D_V$. This construction can also be used for additional aggregation functionality, e.g., count queries can be supported by encrypting value attributes fixed to one. The construction described before can be extended canonically to support multiple value attributes by repeated application for each value column. However, the group membership is then encoded in each value separately, resulting in redundant information. In order to achieve better storage efficiency, we store the value attributes and the shift values $s(g_i)$ determined by the group value $g_i$ separately and multiply them on the server when queried. Since these values are assumed to be sensitive, they have to be stored encrypted in a way still supporting multiplication (over ciphertexts). This can be realized using somewhat homomorphic encryption (SWHE) $\Sigma$ = (Gen, Enc, Dec, $\oplus$, $\otimes$) supporting one single multiplication and being additively homomorphic even after multiplication [4]. We use the notation $[\![x]\!]^{\otimes}_{\oplus}$ to describe the encryption of a plaintext $x$ using algorithm Enc of a SWHE scheme. The operations $\oplus$ and $\otimes$ denote the additively homomorphic and multiplicatively homomorphic operation. We denote the multiplication of a ciphertext $[\![x]\!]^{\otimes}_{\oplus}$ by a plaintext $y$ as $[\![x]\!]^{\otimes}_{\oplus} \otimes y = [\![x \cdot y]\!]^{\otimes}_{\oplus}$. This operation uses the additively homomorphic property of the encryption scheme only and should not be confused with the multiplicatively homomorphic operation that multiplies two ciphertexts.

Given the previous example in Table 1, the transformed table using dynamically shifted bucketization is given in Table 3 with $s(\text{male}) = 1$ and $s(\text{female}) = 2^{32}$.

| E_Salary | E_Gender | ... |
|---|---|---|
| $[\![1000]\!]^{\otimes}_{\oplus}$ | $[\![1]\!]^{\otimes}_{\oplus}$ | |
| $[\![5000]\!]^{\otimes}_{\oplus}$ | $[\![2^{32}]\!]^{\otimes}_{\oplus}$ | |
| $[\![1500]\!]^{\otimes}_{\oplus}$ | $[\![2^{32}]\!]^{\otimes}_{\oplus}$ | |
| $[\![3000]\!]^{\otimes}_{\oplus}$ | $[\![1]\!]^{\otimes}_{\oplus}$ | |
| $[\![2000]\!]^{\otimes}_{\oplus}$ | $[\![1]\!]^{\otimes}_{\oplus}$ | |

**Table 3: Encrypted table with dynamic shifts.**

To generalize the construction for multiple attributes, we evaluate the shift $s(g_i)$ of a group attribute value $g_i$ on the server described as polynomials instead of storing it: determine the coefficients of the polynomial $P(X) = \sum_{i=0}^{B-1} a_i \cdot X^i$ such that $P(x) = (|D_V|)^x = s(g_i)$ for $x \in \{0, \ldots, B-1\}$ given by $x = f(g_i) \bmod B$ with mapping function $f$. Recall, that encryptions of a somewhat homomorphic encryption scheme

can be multiplied by a plaintext value. Thus, the server can evaluate polynomials given plaintext coefficients $a_i$ of $P$ and encrypted monomials $[\![1]\!]^{\otimes}_{\oplus}$, $[\![x]\!]^{\otimes}_{\oplus}$, $[\![x^2]\!]^{\otimes}_{\oplus}$, …, $[\![x^{B-1}]\!]^{\otimes}_{\oplus}$, where the degree of this polynomial increases linearly with the bucket size $B$. We transfer the coefficients to the server during encryption of the database to reduce the network bandwidth of aggregation queries.

## 3.4 Multiple Grouping Attributes

Based on the previous ideas we describe our final SAGMA construction supporting GroupBy statements with multiple grouping attributes in the same query as stated in Section 4. Referring to the example from Table 1 this construction supports statements grouping by "Gender" and "Department" or arbitrary subsets thereof, e.g., solely GroupBy "Department".

*Naïve scheme.* While inefficient, this can be implemented using the previous scheme with the power set of group attributes. Note, that it is necessary to use a combined bucket size of $B^i$ for a subset of $i$ attributes in order to avoid additional leakage. Assuming two attributes and the bucket size $B = 2$, we demonstrate a potential attack where an adversary can query a GroupBy operation for both attributes separately and their combination. These queries leak the bucket membership for both attributes individually and the combination, as illustrated in Table 4. Based on the two individual attributes, the two rows are indistinguishable since they are part of the same buckets. However, the two rows do not contain the same values, thus they might be mapped to separate buckets of the combined attribute. Mapping them to separate buckets leaks the fact that these two rows do not contain the same values. Both buckets $\text{Gen}_1$ and $\text{Dept}_1$ contain two values, therefore there are four possible value combinations of the two buckets. To prevent this leakage, all value combinations of the two buckets are mapped to the same bucket of the combined attribute requiring a bucket size of $B = 4$. More generally, an attribute combination of $i$ attributes requires a bucket size of $B^i$ to achieve the same leakage as in the single attribute case.

| ID | Gender | Department | Gender, Department |
|---|---|---|---|
| 1 | $\text{Gen}_1$ | $\text{Dept}_1$ | $\text{GenDept}_1$ |
| 2 | $\text{Gen}_1$ | $\text{Dept}_1$ | $\text{GenDept}_2$ |

**Table 4: Possible bucket memberships for Table 1.**

*Improved scheme.* We use the polynomial approach described in Section 3.3 and extend it to multivariate polynomials where one variable represents one group attribute. The shift for a combination of attributes $G_1, \ldots, G_l$ can be determined by the polynomial of $l$ variables:

$$P(G_1, \ldots, G_l) = \sum_{i_1,\ldots,i_l} a_{i_1,\ldots,i_l} \cdot G_1^{i_1} \cdots G_l^{i_l}.$$

We improve the naïve scheme by reusing the monomials required for individual attribute grouping for the grouping of attribute combinations. More generally, to group a set of attributes all monomials of the subsets can be used, thus reducing the number of monomials required to be stored on the server. Due to the empty product, only $B - 1$ monomials are necessary for a single attribute using bucket size $B$.

Considering three group attributes G1, G2 and G3 we demonstrate the difference. The naïve scheme requires one monomial for each of the individual attributes G1, G2 and G3, three monomials for the attribute combinations of size two and seven monomials for the combination of all three attributes. The improved scheme also requires one monomial for each individual attribute; however, the attribute combinations of size two only require one additional monomial because the two monomials of the individual attributes can be reused. The idea of monomial reuse is sketched in Figure 2; here the beginning of an arrow represents monomials required to support grouping by a specific attribute combination where monomials can be re-used for the attribute combination denoted at the arrowhead. This relation is transitive, i.e. the attribute combination of all three attributes can reuse the monomials of all its subsets, thus only requiring one additional monomial. In this case, we can reduce the required number of monomials from nineteen to seven.

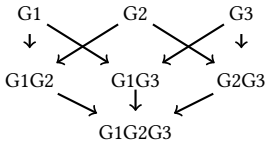



**Figure 2: Monomial reuse that supports multiple attributes.**

Based on Table 1 we give an example that demonstrates grouping of multiple attributes. In this example, we assume bucket size $B = 2$ resulting in the following buckets: one bucket for the "Gender" attribute named $\text{Gen}_1$ containing {male, female} and two buckets for the "Department" attribute named $\text{Dept}_1$ containing {Sales, Finance} and $\text{Dept}_2$ containing {Facility}. Bucket membership for each row is indexed using searchable symmetric encryption as sketched [1] in Table 5.

[1] This is a sketch for demonstration purpose only and constructing efficient SSE indexes is its own line of research.

For readers unfamiliar with searchable symmetric encryption (SSE) we give a high-level summary of its functionality but omit technical details.

Definition 1. *A searchable symmetric encryption scheme consists of the following four algorithms:*

- $K_{SSE} \leftarrow \text{Gen}_{SSE}(1^\lambda)$**:** *Generates a private key.*
- $I \leftarrow \text{Enc}_{SSE}(K_{SSE}, D)$**:** *Creates an encrypted index from a document collection. A document collection is a collection of documents identified by unique document identifiers and consisting of multiple keywords.*
- $t_w \leftarrow \text{Token}_{SSE}(K_{SSE}, w)$**:** *Creates a token $t$ for a keyword $w$ using the private key.*
- $D(w) \leftarrow \text{Search}_{SSE}(I, t_w)$**:** *Uses token $t_w$ and an encrypted index $I$ to search for documents containing the keyword $w$. Returns the document identifiers of all matching documents.*

In our use case for SSE, the encrypted index is created for all bucket identifiers of the complete table consisting of multiple rows. Particularly, the document collection $D$ then corresponds to the complete table and one document corresponds to one specific row, where each such "document" contains its specific bucket identifiers as searchable keywords.

Using SSE, rows belonging to a specific bucket can only be determined using a token generated by the client. Assume mapping functions $f_1(\text{male}) = 0$, $f_1(\text{female}) = 1$ and $f_2(\text{Sales}) = 0$, $f_2(\text{Finance}) = 1$ and $f_2(\text{Facility}) = 2$. Note that each value is reduced mod $B$ before encryption. Further, in order to support grouping by both attributes Gender and Department, the client must generate and outsource an additional monomial, namely their product. This results in the encrypted and outsourced data sketched in Table 6.

| Bucket | Rows |
|---|---|
| $\text{Gen}_1$ | 1, 2, 3, 4, 5 |
| $\text{Dept}_1$ | 1, 2, 3, 4 |
| $\text{Dept}_2$ | 5 |

**Table 5: Bucket index supporting multiple attributes.**

| ID | E_Salary | E_Gender | E_Department | E_Gender · Dept |
|---|---|---|---|---|
| 1 | $[\![1000]\!]^{\otimes}_{\oplus}$ | $[\![0]\!]^{\otimes}_{\oplus}$ | $[\![0]\!]^{\otimes}_{\oplus}$ | $[\![0]\!]^{\otimes}_{\oplus}$ |
| 2 | $[\![5000]\!]^{\otimes}_{\oplus}$ | $[\![1]\!]^{\otimes}_{\oplus}$ | $[\![0]\!]^{\otimes}_{\oplus}$ | $[\![0]\!]^{\otimes}_{\oplus}$ |
| 3 | $[\![1500]\!]^{\otimes}_{\oplus}$ | $[\![1]\!]^{\otimes}_{\oplus}$ | $[\![1]\!]^{\otimes}_{\oplus}$ | $[\![1]\!]^{\otimes}_{\oplus}$ |
| 4 | $[\![3000]\!]^{\otimes}_{\oplus}$ | $[\![0]\!]^{\otimes}_{\oplus}$ | $[\![0]\!]^{\otimes}_{\oplus}$ | $[\![0]\!]^{\otimes}_{\oplus}$ |
| 5 | $[\![2000]\!]^{\otimes}_{\oplus}$ | $[\![0]\!]^{\otimes}_{\oplus}$ | $[\![0]\!]^{\otimes}_{\oplus}$ | $[\![0]\!]^{\otimes}_{\oplus}$ |

**Table 6: Encrypted table with multiple attributes.**

We determine a multi-variate polynomial $P(G_1, G_2)$ that maps combinations of attribute values to the proper shift, using a system of linear equations. In the following, $G_1$ represents the attribute Gender and $G_2$ the attribute Department. However, the solution of the linear system can be reused for other attributes using the same bucket size.

$$\begin{matrix} 1 & G_1 & G_2 & G_1 \cdot G_2 \end{matrix}$$
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2^{32} \\ 2^{64} \\ 2^{96} \end{pmatrix}$$

One solution for the system is $a_0 = 1, a_1 = 2^{64}-1, a_2 = 2^{32}-1$, $a_3 = 2^{96} - 1 - (2^{64} - 1) - (2^{32} - 1)$. These coefficients are transferred to the server in plaintext during the encryption phase. Later, the client wishes to execute the query from Listing 2 grouping by both attributes.

```
SELECT SUM(Salary), Gender, Department
FROM example GROUP BY Gender, Department;
```

**Listing 2: Query grouping by two attributes.**

Therefore, the client determines SSE tokens for the buckets $\text{Gen}_1$, $\text{Dept}_1$ and $\text{Dept}_2$ to send them to the server in addition to the identifier of the attribute Salary to be aggregated and the identifiers of the attributes Gender and Department to be grouped. The server uses these tokens to determine the rows that belong to a specific bucket. Then, by calculating the intersection, the server determines the rows that belong to a specific bucket combination. Alternatively, an SSE scheme that supports Boolean queries can be used to determine joint bucket membership without leaking the bucket membership of individual attributes.

In our example, the first four rows belong to the same bucket combination $(\text{Gen}_1, \text{Dept}_1)$, while the last row with ID 5 belongs to bucket combination $(\text{Gen}_1, \text{Dept}_2)$. The server determines the (encrypted) shift for each row by evaluating the polynomial[2] $P$ on the encrypted grouping monomials $P(G_1, G_2) = a_0 + a_1 \cdot G_1 + a_2 \cdot G_2 + a_3 \cdot G_1 \cdot G_2$. For the first row, the polynomial evaluates to 1, for the second row to $2^{64}$ and so on. Notice that the result of the evaluation is encrypted, hence hiding the shift value in each bucket. Multiplication of the corresponding entry in the value column Salary by the encrypted shift yields the shifted value to be aggregated. The server adds up all the shifted values for each bucket combination separately and returns the result. By decrypting the result, the client receives the packed plaintexts depicted in Table 3. Since the client has chosen the shifts of all value combinations by determining the polynomial's coefficients, the client can map each part of the plaintext to a group attribute combination. Table 7 shows the final result of the

[2]This is possible without calling $\otimes$ since the polynomial's coefficients are outsourced in plaintext.

| | | | |
|---|---|---|---|
| 1500 | 5000 | 0 | 4000 |

**(a) Bucket combination ($\mathbf{Gen}_1, \mathbf{Dept}_1$)**

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 2000 |

**(b) Bucket combination ($\mathbf{Gen}_1, \mathbf{Dept}_2$)**

**Figure 3: Sketch of decrypted aggregation result for different buckets.**

query. Note that the coefficients can be reused for other attributes if they use the same bucket size.

| SUM(Salary) | Gender | Department |
|---|---|---|
| 4000 | male | Sales |
| 5000 | female | Sales |
| 1500 | female | Finance |
| 2000 | male | Facility |

**Table 7: Result of query stated in Listing 2 on Table 1.**

## 4 FORMALIZATION

Before we give a comprehensive formal description, we define the interface of the SAGMA construction for secure aggregation of $k$ aggregation values and supporting a combination of up to $t$ arbitrary grouping attributes in one query. SAGMA consists of the following six possibly probabilistic polynomial time (PPT) algorithms:

- $(pp, K) \leftarrow \text{Setup}(1^\lambda, D_1, \ldots, D_l)$ : Executed on the client. Generates the cryptographic keys and outputs the public parameters $pp$ and the secret key $K$. $D_1, \ldots, D_l$ denote the value domains of the grouping attributes.
- $C \leftarrow \text{EncTable}(K, \{\{v_{i,j}\}_{j=1}^{k}, \{g_{i,j}\}_{j=1}^{l}\}_{i=1}^{rows})$ : Executed on the client. Using the given cryptographic keys it encrypts the table executing EncRow for each row. Depending on the concrete construction, an index for grouping or additional filtering is created. The result is then outsourced to the untrusted server.
- $c \leftarrow \text{EncRow}(pk, v_1, \ldots, v_k, g_1, \ldots, g_l)$ : Is executed on the client for each database row. Encrypts the row. This operation enables the client to add subsequent rows to the initially encrypted table.
- $t_{grp} \leftarrow \text{AggGrpByToken}(K, V, Q)$ : Executed on the client. Creates a grouping token to execute grouping by up to $t$ grouping columns in $Q$ and aggregation of value columns in $V$.
- $c_{agg} \leftarrow \text{AggGrpBy}(pp, t_{grp}, C)$ : Executed on the server. Aggregates the encrypted data with GroupBy statement using the tokens generated in the previous step.
- $\text{res} \leftarrow \text{DecAgg}(K, c_{agg})$ : Executed on the client, decrypts the encrypted result of the queried aggregation with GroupBy statement.

We refer to Table 8 for an overview of the used variables in our algorithm descriptions. Founded on the formalization,

| Variable | Description |
|---|---|
| $D_i$ | Value domain for $i$-th grouping attribute |
| $l$ | Number of grouping attributes |
| $\lambda$ | Security parameter |
| $f_i$ | Pseudorandom function mapping grouping attribute values of column $i$ to natural numbers |
| $a_i$ | Polynomial coefficients used for oblivious shift calculation |
| $B$ | Bucket size of each bucket containing $B$ group attribute values |
| $pp$ | Public parameters |
| $K_{SSE}$ | Secret key for searchable encryption scheme |
| $K$ | Secret master key |
| $\|S\|$ | Number of elements in set $S$ |
| $v_{i,j}$ | Attribute value for the $j$-th value column (in the $i$-th row if stated) |
| $g_{i,j}$ | Attribute value for the $j$-th grouping column (in the $i$-th row if stated) |
| $k$ | Number of value attributes |
| $D$ | Document collection for SSE containing the bucket identifiers of all rows |
| $I$ | Searchable encrypted index for bucket memberships of all grouping attribute values for each row |
| $r_{i,j}$ | Offset of value in bucket for $j$-th grouping column in $i$-th row |
| $C$ | Encrypted database |
| $m_i$ | Monomial for oblivious shift calculation of bucketized group values |
| $Q$ | GroupBy clause of aggregation query for up to $t$ grouping attributes |
| $s_i$ | Number of buckets for $i$-th grouping attribute |
| $t_{i,j}$ | SSE token for the $j$-th bucket of the $i$-th grouping attribute |
| $p$ | Size of joint bucket |
| $R$ | Rows for aggregation of joint bucket |
| $S_i$ | Shift for the $i$-th row |

**Table 8: Overview of used variables.**

we discuss storage efficiency in Section 4.1, state a theoretical upper bound for the information leakage of our scheme in Section 4.2 and compare our construction with the approach based on pre-computation and Seabed in Section 6.2.

Let $\Sigma$ = (Gen, Enc, Dec, $\oplus$, $\otimes$) be a semantically secure SWHE scheme supporting one single multiplication as introduced before. Let $SSE = (\text{Gen}_{SSE}, \text{Enc}_{SSE}, \text{Token}_{SSE}, \text{Search}_{SSE})$ be an adaptively semantically secure SSE scheme as stated in Definition 1. In the following, we give an intuition for each SAGMA algorithm together with a formal description in Algorithms 1– 6.

**Algorithm 1** Key generation algorithm.

**Setup**$(1^{\lambda_0}, 1^{\lambda_1}, D_1, \ldots, D_l)$:
$(pk, sk) \leftarrow \text{Gen}(1^{\lambda_0})$
$F = (f_1, \ldots, f_l),\ f_i : D_i \mapsto \{0, \ldots, |D_i|\}$
$pp = (pk, a_0, \ldots, a_{B^l-1})$
$K_{SSE} \leftarrow \text{Gen}_{SSE}(1^{\lambda_1})$
**return** $pp, K = (pk, sk, F, K_{SSE})$

*Setup* as stated in Algorithm 1 generates the cryptographic keys, defines the mapping functions for the group domains and calculates the polynomial determining the shift values.

*EncTable* as stated in Algorithm 2 creates and indexes the group buckets for all grouping attributes using SSE. Note that EncRow is called for a set of rows encrypting aggregation values and bucketized group values.

**Algorithm 2** Encryption algorithm for the complete table.

**EncTable**$(K,\ \{\{v_{i,j}\}_{j=1}^{k}, \{g_{i,j}\}_{j=1}^{l}\}_{i=1}^{rows})$:
$D = \{\{j : \frac{f_j(g_{i,j})}{B}\}_{j=1}^{l}\}_{i=1}^{rows}$
$I \leftarrow \text{Enc}_{SSE}(K_{SSE}, D)$
$r_{i,j} = f_j(g_{i,j}) \bmod B$
$c_i \leftarrow \text{EncRow}(pk, v_{i,1}, \ldots, v_{i,k}, r_{i,1}, \ldots, r_{i,l})$
**return** $I, C = c_1, \ldots, c_{rows}$

Particularly, *EncRow* as stated in Algorithm 3 determines monomials of the bucketized group values required to determine appropriate shifts on the server later on and applies somewhat homomorphic encryption to the aggregation values and the monomials to support dynamic shifting. Note that this algorithm can be used for database updates after the initial table encryption if the bucket index $I$ is updated correspondingly.

**Algorithm 3** Encryption algorithm for a single row.

**EncRow**$(pk, v_1, \ldots, v_k, g_1, \ldots, g_l)$:
$$m_1, \ldots, m_{B^l} = \left\{ \prod_{i=1}^{l} g_i^{e_i} \mid \forall\ 0 \le e_i < B \right\}$$
**return** $c = (\llbracket v_1 \rrbracket_{\oplus}^{\otimes}, \ldots, \llbracket v_k \rrbracket_{\oplus}^{\otimes}, \llbracket m_1 \rrbracket_{\oplus}^{\otimes}, \ldots, \llbracket m_{B^l} \rrbracket_{\oplus}^{\otimes})$

*AggGrpByToken* as stated in Algorithm 4 creates SSE search tokens for all buckets of the grouping attributes and outputs the identifier of the attribute to aggregate, the identifiers of the group attributes and the SSE tokens.

**Algorithm 4** Group token generation algorithm.

**AggGrpByToken**$(K, V \in \{1, \ldots, k\}, Q \subseteq \{1, \ldots, l\})$:
**for all** $q \in Q$ **do**
$s_q = \left\lceil \frac{|D_q|}{B} \right\rceil$
**for all** $1 \le b \le s_q$ **do**
$t_{q,b} \leftarrow \text{Token}_{SSE}(K_{SSE}, q : b)$
**return** $V, Q, \{t_{q,b} \mid q \in Q, b \in \{1, \ldots, s_q\}\}$

*AggGrpBy* as stated in Algorithm 5 uses the encrypted index to determine the rows of all buckets of the group attributes. By calculating the intersection, rows belonging to joint buckets are determined and aggregation is executed for each joint bucket. Aggregation involves determining the appropriate shift for each row and multiplying the shift by the value attribute. Finally, one encrypted result is returned for each joint bucket.

**Algorithm 5** Aggregation algorithm over encrypted data.

**AggGrpBy**$(pp, V, Q, \{t_{q,1}, \ldots, t_{q,s_q} \mid q \in Q\}, C, I)$:
$s_q = \left\lceil \frac{|D_q|}{B} \right\rceil$, for $q \in Q$
$p = B^{|Q|} - 1$
determine coefficients $a_0, \ldots, a_p$ for $Q$
determine indices $i_1, \ldots, i_p$ for $Q$
**for all** $(b_1, \ldots, b_{|Q|}), b_i \in \{1, \ldots, s_{Q_i}\}$ [3] **do**
$R \leftarrow \bigcap_{i=1}^{|Q|} \text{Search}_{SSE}(I, t_{Q_i, b_i})$
**for all** $r \in R$ **do**
$S_r = \llbracket a_0 \rrbracket_{\oplus}^{\otimes} \oplus \bigoplus_{j=1}^{p} a_j \otimes \llbracket m_{i_j} \rrbracket_{\oplus}^{\otimes}$
$agg_{b_1, \ldots, b_{|Q|}} = \bigoplus_{r \in R} \llbracket v_V \rrbracket_{\oplus}^{\otimes} \otimes S_r$
**return** all aggregates $agg_{b_1, \ldots, b_{|Q|}}$

*DecAgg* as stated in Algorithm 6 decrypts and unpacks the packed ciphertexts that result from the aggregation.

**Algorithm 6** Decryption algorithm for query result.

**DecAgg**$(K, agg_1, \ldots, agg_s)$:
**for** $1 \le i \le s$ **do**
$u_{i,1}, \ldots, u_{i,B^{|Q|}} \leftarrow \text{Dec}(sk, agg_i)$
**return** $u_{1,1}, \ldots, u_{s,B^{|Q|}}$

[3] Enumerates joint buckets using the cartesian product of the buckets of the queried attributes.

## 4.1 Efficiency

By precomputing $B^l - 1$ monomials for the polynomial evaluation, our construction requires a SWHE scheme supporting only one single multiplication. On the other hand, storing the monomials limits the bucket size $B$ and the number of group attributes $l$.

Particularly, our SAGMA construction requires $B^l - 1$ monomials with exponential increase in the number of group attributes $l$. Generally, only small subsets of group attributes occur together in one query. Limiting the number of group attributes stated in one query allows us to reduce the number of required monomials. More particular, assume the number of attributes is limited by a constant $t$. We denote the number of monomials that have to be stored for each row in a database with $l$ group attributes by $m(l, t)$. We can apply our naïve construction to all subsets of attributes of size $t$, which requires to store $m(l,t)_{\text{naive}} = \binom{l}{t} \cdot (B^t - 1) \leq \left(\frac{l \cdot e}{t}\right)^t \cdot (B^t - 1)$ monomials per row. As a result, the number of required monomials is bounded polynomially, instead of exponentially in $l$. However, the monomial reuse in our improved scheme reduces the number of required monomials further. To calculate the exact number of monomials necessary, we

| t | $\mathbf{m(l,t) - m(l,t-1)}$ |
|---|---|
| 1 | $l \cdot (B-1)$ |
| 2 | $\binom{l}{2} \cdot \left(B^2 - 1 - 2 \cdot (B-1)\right)$ |
| 3 | $\binom{l}{3} \cdot \left(B^3 - 1 - \binom{3}{1} \cdot (B-1) - \binom{3}{2} \cdot (B-1)^2\right)$ |
| ⋮ | ⋮ |
| $t$ | $\binom{l}{t} \cdot \left(B^t - \sum_{i=0}^{t-1} \binom{t}{i} (B-1)^i\right)$ |

**Table 9: Required number of monomials to support grouping up to $k$ attributes.**

examine how many monomials are necessary to support grouping of $t$ attributes, if we already support grouping of $t-1$ attributes. Notice that $m(l, 0) = 0$. The result of this iterative construction is given for some $t$ in Table 9. In the first step, we aim to support grouping of a single attribute. This requires storing $B - 1$ powers of all attributes. To support grouping of two attributes $B^2 - 1$ monomials are required for each subset of attributes of size two ($-1$ because of the empty product). The $B - 1$ powers of the two attributes can be used and thus be subtracted. This can be generalized for an arbitrary $t$ resulting in the following equation:

$$m(l,t) - m(l,t-1) = \binom{l}{t} \cdot \left(B^t - \sum_{i=0}^{t-1} \binom{t}{i} (B-1)^i\right) \overset{*}{=} \binom{l}{t} \cdot (B-1)^t .$$

The above transformation $(*)$ can be verified using a proof by induction. More precisely, we define the number of monomials required to support grouping of $t$ attributes in one query given a database with $l$ possible group attributes as:

$$m(l,t) = \sum_{i=1}^{t} m(l,i) - m(l,i-1) = \sum_{i=1}^{t} \binom{l}{i} \cdot (B-1)^i .$$

A lower bound of this number grows polynomially in $l$ and thus $m(l, t) \in \Theta(l^t \cdot B^t)$:

$$\sum_{i=1}^{t} \binom{l}{i} \cdot (B-1)^i \geq \binom{l}{t} \cdot (B-1)^t \geq \left(\frac{l}{t}\right)^t \cdot (B-1)^t .$$

## 4.2 Security

Given a semantically secure SWHE encryption scheme $\Sigma$ and an adaptively semantically secure SSE scheme, the construction as stated in Algorithms 1– 6 essentially leaks the bucket membership of all queried group attributes. We emphasize that our construction offers improved security compared to the encryption of individual group attributes either deterministically or searchable. While deterministic encryption as well as searchable encryption leaks the frequencies of the whole plaintext domain after the execution of one aggregation query, our construction only leaks the frequencies of distinct buckets.

For a formal security analysis, we use a simulation-based security definition following the work on searchable symmetric encryption by Curtmola et al. [9]. Basically, simulation-based security definitions consist of two experiments, the real and the simulated experiment. The real experiment describes a regular protocol sequence, i.e., in our case an adversary $\mathcal{A}$ chooses a plaintext database table to be encrypted and queries the grouping tokens to be created. Encryption of the database table and generation of grouping tokens is executed by the regular algorithms of the scheme. In contrast, in the simulated experiment the encrypted database and the grouping tokens are created by a simulator $\mathcal{S}$ that only has access to limited information about the plaintext database and the queries. This limited information is modeled by a leakage function $\mathcal{L}$, whose output is given to the simulator. Both experiments output the encrypted database and the created grouping tokens. Intuitively, a scheme is secure if the output of both experiments is computationally indistinguishable, i.e. no PPT algorithm exists that has non-negligible advantage in distinguishing the two distributions. A formal description of these two experiments is given in Figure 4. The security notion is stated in Definition 2. Notice, that our security definition is adaptive, i.e. the adversary has access to the grouping tokens created for earlier queries and can choose the next query in dependence on the tokens. In contrast, non-adaptive security definitions require the adversary to choose

all queries at once, which is a weaker security definition and a far less realistic scenario for applications.

Definition 2 (Adaptive $\mathcal{L}$-Security). *The SAGMA scheme as defined in Section 4, is adaptively $\mathcal{L}$-secure, if for all PPT adversaries $\mathcal{A} = (\mathcal{A}_0, \ldots, \mathcal{A}_{q+1})$, there exists a PPT simulator $\mathcal{S} = (\mathcal{S}_0, \ldots, \mathcal{S}_{q+1})$ such that for all PPT algorithms $\mathcal{D}$ it holds that*

$$\Big|\Pr\left[\mathcal{D}(v, st_{\mathcal{A}}) = 1 : (v, st_{\mathcal{A}}) \leftarrow \mathbf{Real}^*{}_{\mathcal{A}}(\lambda)\right] - \Pr\left[\mathcal{D}(v, st_{\mathcal{A}}) = 1 : (v, st_{\mathcal{A}}) \leftarrow \mathbf{Sim}^*{}_{\mathcal{A},\mathcal{S}}(\lambda)\right]\Big| \leq \mathrm{negl}(\lambda)$$

*where $q = \mathrm{poly}(\lambda)$ and $\mathrm{negl}(\lambda)$ is negligible in $\lambda$.*

*Formalized Information Leakage.* We construct a simulator $\mathcal{S}$ fulfilling Definition 2. $\mathcal{S}$ has restricted information in form of the identifiers of the value attribute and the group attributes $(V, Q)$ of all queries. Both $V$ and $Q$ are only identifiers of the queried attributes and do not contain the actual values of these attributes, i.e., they identify the column to be aggregated and to be grouped by. Further, the trace $\tau$, that is the information leaked by SSE is forwarded to $\mathcal{S}$, hence the overall leakage we proof as upper bound is formalized as:

$$\mathcal{L}\left(T, (V_1, Q_1) \ldots, (V_i, Q_i)\right) = \left((V_1, Q_1), \ldots, (V_i, Q_i), \tau_i\right).$$

We do not explicitly mention the number of rows and columns of the database and the bucket size $B$ in the leakage but assume them to be public. Notice that each aggregation query involves multiple SSE queries, one for each bucket of the queried group attributes. Thus, the trace after $i$ aggregation queries contains the access and the search pattern of $i' \geq i$ keyword queries, namely $\tau_i = \tau(D, w_1, \ldots, w_{i'})$. The trace itself consists of the sizes of the documents and the access pattern (i.e. the document identifiers of matching documents) and the search pattern that discloses if two tokens correspond to the same keyword. In our case, the search pattern corresponds to a bucket identifier and the access pattern reveals the rows contained in each bucket. In summary, the overall leakage of our SAGMA construction can be described as follows: the identifiers of the grouping attributes are leaked and for each queried group attribute, the construction leaks the mapping of rows to bucket identifiers, which is included in the access pattern of SSE.

We use SSE as black box, including its common security definition as introduced by Curtmola et al. [9].

Theorem 1. *If the used SSE scheme is adaptively semantically secure, and the used SWHE scheme $\Sigma$ is semantically secure, then our SAGMA construction as stated in Algorithms 1–6 is adaptively semantically $\mathcal{L}$-secure according to Definition 2.*

Proof. For our proof sketch we give an intuition why a simulator exists such that the outputs of the experiments $\mathbf{Real}^*{}_{\mathcal{A}}$ and $\mathbf{Sim}^*{}_{\mathcal{A},\mathcal{S}}$ are computationally indistinguishable for every possible PPT adversary $\mathcal{A}$. Using a standard hybrid argument, we show that our construction satisfies the security definition. The output of both experiments has the same structure, that is, $pp, C' = (C, I), t_{grp,1}, \ldots, t_{grp,q}, st_{\mathcal{A}}$. Since the public parameters $pp$ have been created by the same algorithm Setup, their distribution is identical. If a PPT algorithm exists that distinguishes the encrypted database $C$ of the two experiments, then an adversary can be constructed breaking semantic security of $\Sigma$ contradicting our assumption. Similarly, if the encrypted index $I$ or the SSE tokens contained in the grouping tokens $t_{grp}$ can be distinguished, then the security of the SSE scheme can be broken. Furthermore, the grouping tokens $t_{grp}$ contain the identifiers of the queried value attribute and group attributes. Since these identifiers are contained in the leakage, they are identical in both experiments.

Because no single component of the output can be distinguished by a PPT algorithm, using the hybrid argument it follows that the outputs of both experiments are indistinguishable and thus our construction fulfills the security definition. □

$\mathbf{Real}^*{}_{\mathcal{A}}(\lambda)$:
$D_1, \ldots, D_l, st_{\mathcal{A}} \leftarrow \mathcal{A}_0(1^\lambda)$
$pp, K \leftarrow \mathrm{Setup}(1^\lambda, D_1, \ldots, D_l)$
$T, st_{\mathcal{A}} \leftarrow \mathcal{A}_1(st_{\mathcal{A}}, pp)$
$C \leftarrow \mathrm{EncTable}(K, T)$
**for** $1 \leq i \leq q$ **do**
$Q_i, st_{\mathcal{A}} \leftarrow \mathcal{A}_{i+1}(st_{\mathcal{A}}, pp, C, t_{grp,1}, \ldots, t_{grp,i-1})$
$t_{grp,i} \leftarrow \mathrm{AggGrpByToken}(K, (V_i, Q_i))$
**return** $pp, C, t_{grp,1}, \ldots, t_{grp,q}, st_{\mathcal{A}}$

$\mathbf{Sim}^*{}_{\mathcal{A},\mathcal{S}}(\lambda)$:
$D_1, \ldots, D_l, st_{\mathcal{A}} \leftarrow \mathcal{A}_0(1^\lambda)$
$pp, st_{\mathcal{S}} \leftarrow \mathcal{S}_0(1^\lambda, D_1, \ldots, D_l)$
$T, st_{\mathcal{A}} \leftarrow \mathcal{A}_1(st_{\mathcal{A}}, pp)$
$C, st_{\mathcal{S}} \leftarrow \mathcal{S}_1(st_{\mathcal{S}}, \mathcal{L}(T))$
**for** $1 \leq i \leq q$ **do**
$Q_i, st_{\mathcal{A}} \leftarrow \mathcal{A}_{i+1}(st_{\mathcal{A}}, pp, C, t_{grp,1}, \ldots, t_{grp,i-1})$
$t_{grp,i}, st_{\mathcal{S}} \leftarrow \mathcal{S}_i(st_{\mathcal{S}}, \mathcal{L}(T, (V_1, Q_1), \ldots, V_i, Q_i))$
**return** $pp, C, t_{grp,1}, \ldots, t_{grp,q}, st_{\mathcal{A}}$

**Figure 4: Security Experiments – Real vs. Ideal**

Rows that are part of different buckets can be distinguished using the access pattern; however, the underlying group values for each row cannot be extracted from this leakage. More precisely, distinct group values that are mapped into the same bucket are indistinguishable for any adversary; an adversary successfully distinguishing these rows with non-negligible probability can break the somewhat homomorphic

encryption scheme, i.e. distinguish encrypted monomials. Given a pseudorandom function that maps individual group values to buckets, an adversary cannot tell which group value is mapped into which bucket for group values with the same frequency. Thus, the provided security heavily depends on the characteristics of the underlying plaintext values – we emphasize, that this is true for all searchable symmetric encryption schemes. As already highlighted by the designers of Seabed [25] the best security can be achieved by adding dummy rows to each bucket until the same value frequency $t$ is reached for each bucket hence making all buckets indistinguishable. However, a naïve implementation of such dummy rows would result in best security but induce high computational overhead. Specifically, assuming bucket size $B$ and the $B$ most-frequent group values occur $r$ times in total then $t$ is set to $r$. Assuming all other group values occur exactly once, each bucket is filled up with $t-B$ dummy values. The number of different buckets is bounded by $\lceil \frac{|D|}{B} \rceil$, thus the number of total dummy values is bounded by $(\lceil \frac{|D|}{B} \rceil - 1) \cdot (t - B)$. In addition to dummy values we describe further mechanisms, such as optimizing the mapping function and splitting group attribute values in the next section. These can be used to optimize the total overhead for a targeted security level. We emphasize, that these protection mechanisms are entirely data-dependent and their effect should be analyzed for individual use-cases.

## 5 BUCKET PARTITIONING

The choice of the mapping function directly influences the bucket distribution and hence information leakage. For instance, consider bucket size $B = 2$ and three group values $g_1, g_2$ and $g_3$ with frequencies 1, 2 and 3. Mapping $g_1$ and $g_3$ into the same bucket results in bucket frequency 4 and $g_2$ is mapped into a separate bucket with bucket frequency 2. An attacker aware of the group attribute distribution can reconstruct the mapping since the bucket frequencies are unique, i.e. they can only be generated by one mapping. In contrast, mapping $g_1$ and $g_2$ into one bucket and $g_3$ into a separate one, results in two buckets with the same frequency. Thus, the indistinguishability of values can be extended beyond values of the same bucket by proper partitioning.

Ceselli et al. [7] examine the use of hash functions for partitioning. They examine selection queries and introduce the exposure coefficient, which quantifies the average probability of correctly guessing the attribute value of an encrypted row based on auxiliary information. The exposure coefficient depends on several factors, e.g., the number of mapping functions resulting in the same bucket distribution, the number of attribute values and the number of buckets with the same frequency, the individual value distribution inside each bucket. In order to reduce exposure, they propose a larger collision factor of the hash function corresponding to larger bucket sizes in our scheme. Founded on the work by Caselli et al. we propose the following additional security mechanisms that increase security compared to deterministic encryption.

(1) Optimal choice of the mapping function, i.e. bucket partitioning with minimal exposure.
(2) Supplementing dummy rows and thus altering the number of values in buckets.
(3) Attribute value splits allow us to partition one group attribute value $g$ with high frequency in two distinct values $g.1$ and $g.2$.

**Optimal mapping function.** We choose a bucket distribution that minimizes exposure. Note, that this approach is limited by the plaintext distribution and the bucket size. For example, given bucket size 2 and three group values $g_1, g_2$ and $g_3$ that occur 1, 2 and 4 times respectively, all possible bucket partitions are distinguishable. Here, a specific choice of the mapping function does not introduce a large computational overhead and can be combined with dummy values and attribute value splits as further protection mechanisms.

**Supplementing dummy values.** The structure of individual buckets can be altered by adding dummy values. Particularly, the number of elements in one bucket can be increased by adding dummy values containing value attributes all set to zero into specific buckets. Thus, they do not influence the aggregation result, but hide the distribution of attribute values. While inserting dummy rows for the use case of Ceselli et al. raises additional client overhead to filter rows, it only increases the server overhead in our use case.

**Attribute value splits.** A group attribute value $g$ can be split into two values $g.1$ and $g.2$ and thereby the frequency is split into two summands. This requires the client to aggregate the sum for $g.1$ and $g.2$ after decryption but modifies value distribution in buckets, hence increases security.

## 6 EVALUATION

We implemented SAGMA described in Section 3.4 using the SWHE scheme published by Boneh, Goh and Nissim (BGN) [4] based on bilinear maps. BGN is the most prominent homomorphic encryption scheme being additively homomorphic and supporting one single multiplication of ciphertexts. We implemented SAGMA in Java using parallelization during query execution and decryption to use multiple cores.

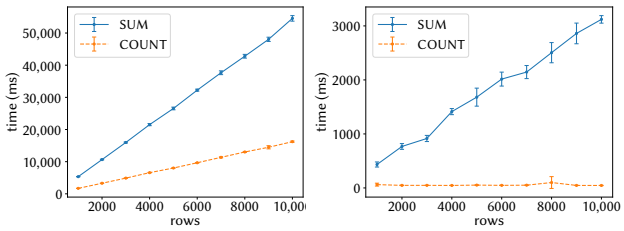
Decryption of BGN requires to calculate the discrete logarithm in a prime-order group, thus the plaintext space has to be restricted to support efficient decryption. Nevertheless, the decryption of a ciphertext with restricted plaintext space of 32 bits takes several seconds on a current laptop and is too inefficient for the evaluation of database aggregations. Hu et al. propose to use the Chinese remainder theorem (CRT) to speed up the decryption of BGN [21]. They split a message

into several parts; homomorphic operations are executed for each part and the result is reconstructed after the decryption of all parts using CRT. This approach offers a trade-off between decryption time on the client and running time of homomorphic operations on the server. Since ciphertext components can only be decrypted if they are small enough and homomorphic operations increase the components, the use of homomorphic operations is limited.

Our implementation supports three different aggregation operations, namely summation, row count and average. The row count can be calculated by aggregating the shifts instead of the shifted values. As the result is limited by the total number of rows the CRT scheme is not required.

## 6.1 Results

We use the *lineitem* table of the TPC-H [4] benchmark to evaluate aggregation time and decryption time. The evaluation runs on a machine with two Intel Xeon E5-2670 CPUs with eight cores each running at 2.60 GHz and 256 GB of RAM. We execute every query ten times and calculate the mean running time, as well as the 95% confidence interval. Network latencies between client and server are not considered. Since performance of SSE has been analyzed extensively before, our implementation uses a plaintext index located on the client to determine the bucket identifiers of rows. We instantiate a cryptographic key with 1024 bits providing about 80 bits of security [2, 12] since BGN is based on the hardness assumption of factorizing a composite modulus.



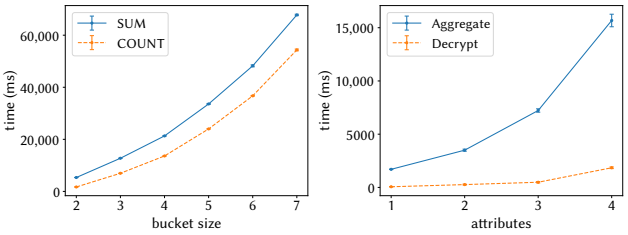

**(a) Aggregation time** **(b) Decryption time**

**Figure 5: Processing time for varying number of rows.**

We evaluate the performance of our SAGMA construction in dependence on the number of rows to be aggregated. Figure 5 shows a linear increase of the aggregation time. Aggregating the count of 1000 rows requires 1.9 s and 10, 000 rows lasts 17.7 s. Summation is less efficient, since it is based on the CRT scheme. The decryption time of the count operation stays constant, while it increases for the summation operation. Again, this is due to the CRT construction limiting the number of supported additions.

[4] See http://www.tpc.org/tpch/.

We emphasize that most queries contain `WHERE` clauses to filter rows which limits the number of rows that have to be aggregated in real use cases making our construction also suitable for larger databases. Such preceding selection is orthogonal to our work and can be implemented efficiently using SSE. Runtime for different bucket sizes $B$ are evaluated




**(a) Varying bucket size** **(b) Varying attribute number**

**Figure 6: Aggregation time for varying bucket size and varying number of combined attributes in one query.**

in Figure 6a. The aggregation time increases superlinearly, summation takes 5.7 s for bucket size 2 and 71.3 s for bucket size 7 due to the use of unit shifts to reduce the size of the CRT components. Instead of one polynomial of degree $B$, $B$ polynomials are required to evaluate the shifts for each row. Hence, the running time increases quadratically in the bucket size. Again, the count operation is more efficient than summation.

Finally, we evaluate the impact of the number of attributes to be grouped. According to Figure 6b, the aggregation time increases superlinearly. Here, the running time increases polynomially in the total number of attributes of the database table if the number of grouping attributes in a single query is limited by $t$. This enables us to store and combine $B^t$ monomials to evaluate the polynomial. Note that this does

| Application | Grouping attributes | | |
|---|---|---|---|
| | 1 | ≤ 2 | ≤ 3 |
| Nextcloud | 100 % | 100 % | 100 % |
| Wordpress | 97 % | 99 % | 100 % |
| Piwik | 25 % | 83 % | 95 % |

**Figure 7: Grouping queries with attribute numbers.**

not limit the applicability of our construction severely. As summarized in Table 7, many applications only group by a small number of attributes in each query as our analysis of popular applications such as Nextcloud, WordPress and Piwik showed. Nextcloud only groups by a single attribute and uses the row count operation exclusively. Similarly, 97% of the grouping queries used by WordPress contain a single group attribute; the largest grouping query contains three

attributes. Finally, Piwik, being an analytics platform, uses grouping functionality extensively. We report that 95% of the grouping queries contain three group attributes or less. The largest grouping query contains five attributes. This indicates that an upper limit of the number of group attributes supported in a single query is no practical restriction.

## 6.2 Comparison

We compare our solution SAGMA with i) pre-computing all results as discussed in Section 2 and ii) Seabed [25], a recent proposal for secure aggregation. We analyze storage requirements on the server and computation efforts required by the client for grouping operations with multiple attributes. We assume a table with $r$ rows, $k$ value attributes, $l$ grouping attributes and denote $t$ as the maximum number of grouping attributes supported in one query. Further, we denote $n$ as the number of additional filtering clauses queried by the client in combination with data aggregation, where the $i$-th filtering clause has a result set size of $\rho_i$. We refer to Table 8 for an overview of all variables used in the following analysis.

We assume the bucket size for SAGMA and the number of common values for Seabed to be equal for all grouping attributes denoted as $B$. For Seabed and SAGMA we denote $C$ as the number of aggregation results for a query with up to $t$ grouping attributes, i.e. $C = |D|^t$. Natively, Seabed does not support grouping by multiple attributes, i.e. aggregating each attribute value combination individually. Thus, we assume all value combinations have been computed on the client and stored on the server.

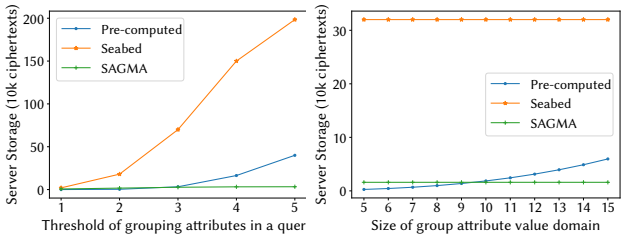
| Scheme | Server Storage | Client |
|---|---|---|
| Pre-computed | $\left(\sum_{i=1}^{t} \binom{l}{i} \cdot \lvert D\rvert^i\right) \cdot k \cdot n$ | 1 |
| Seabed | $\left(\sum_{i=1}^{t} \binom{l}{i} \cdot ((B+1)^i - 1)\right) \cdot k \cdot r$ | $\rho_i \cdot C$ |
| SAGMA | $\left(\left(\sum_{i=1}^{t} \binom{l}{i} \cdot (B-1)^i\right) + k\right) \cdot r$ | $C$ |

**Table 10: Complexity of approaches hiding individual group attribute values. Notation is stated in Table 8.**

In Table 10 we summarize the storage requirements on the server measured in ciphertexts and the required client computation effort for the different approaches. Naïve pre-computation requires each possible query result to be calculated, encrypted by the client and stored on the server. Thus, we enumerate all possible grouping operations combining up to $t$ attributes for storage calculation. The number of distinct combinations for $i$ grouping attributes has size $|D|^i$. Additionally, a different result is stored for each value attribute and for each possible filtering clause. Pre-computing and storing the results for all possible filtering clauses is impractical for real applications, e.g., assuming only one filtering column containing integer values and solely considering equality matching, there are already $2^{32}$ different filtering clauses, increasing exponentially with additional value domains contained in further filtering columns.

In order to support grouping by multiple attributes with Seabed, $(B+1)^i - 1$ columns have to be stored for grouping by $i$ grouping attributes. Again, we enumerate all grouping operations combining up to $t$ attributes. Since Seabed encodes the value attribute in the group attribute column, all columns have to be stored for each value attribute separately. Seabed requires excessive client computation if combined with additional filtering clauses since ASHE is optimized for dense datasets. In the worst case, one operation has to be performed for each row contained in the filtering result for each of the $C$ results. As a result, the client might have to perform more operations than there are rows in the database, making it less efficient than downloading the complete database and aggregating it locally on the client.

SAGMA requires $(B-1)^i$ monomials for a grouping operation with $i$ attributes. As before, we enumerate all grouping operations combining up to $t$ attributes. Value attributes and grouping attributes are stored separately and combined during aggregation. Hence, each value attribute is stored only once. SAGMA is interoperable with searchable encryption for filtering clauses, i.e. filtering clauses are executed on the server and the resulting rows are subsequently aggregated using SAGMA. Client effort is minimal, as the number of different group value combinations is lower bounded by $C$ and for each combination the encrypted aggregation is decrypted by the client. SAGMA is the only scheme that scales with the number of value attributes and the number of possible filtering clauses at the same time. We determine specific values




**(a) Varying parameter $t$** **(b) Varying domain size $|D|$**

**Figure 8: Different server storage requirements of the three schemes: Pre-computed, Seabed and SAGMA.**

for the server storage for $l = 4, t = 3, k = 2, r = 1000, n = 2$ and give the results in Figure 8. We evaluate different values for the maximum number of supported grouping attributes $t$ in Figure 8a, and different group attribute domain sizes $|D|$ in Figure 8b. Seabed requires an excessive amount of storage

| Scheme | Aggregation | Grouping | Security | Proof | Multiple Attributes |
|---|---|---|---|---|---|
| Bucketization [17] | × | ×/✓ | ◐ | × | ×/✓ |
| CryptDB [26] | ✓ | ✓ | ○ | × | ✓ |
| Seabed [25] | ✓ | ✓ | ◐ | ✓ | × |
| This paper: w/o buckets (ref. Section 3.1) | ✓ | ✓ | ● | ✓ | × |
| This paper: SAGMA | ✓ | ✓ | ◐ | ✓ | ✓ |

**Table 11: Comparison of related work.**

space and SAGMA is superior to the pre-computed scheme for $t \geq 3$ and $|D| \geq 10$.

## 7 RELATED WORK

The first approach to group encrypted data uses the concept of *bucketization*; more specifically, the value domains of the attributes are split into non-overlapping buckets that are identified by a unique value [17]. The database is encrypted row-wise and outsourced together with these bucket identifiers. Using the bucket identifiers, the server can group corresponding rows. This bucketization introduces inaccuracy, thus the client refines the grouping result for each bucket after decryption. Here the client exclusively aggregates the grouped values for each bucket after decryption. The proposed scheme requires to execute most of the work on the client, has high communication cost and no security proof is stated. Additional work on bucketization has been published by Hore et al. [19, 20] implementing elementary data security while still providing functionality for range queries on the protected data.

*CryptDB* uses a combination of homomorphic encryption and property-preserving encryption (PPE) to support a wide range of operations for their encrypted database [26]. Adjustable encryption allows the client to unveil specific decryption keys enabling the server to adapt the encryption level of attributes to the requirements of a query. In order to support GroupBy statements, the involved grouping columns are encrypted deterministically leaking the frequencies of the plaintext data. For aggregation queries the authors propose the Paillier cryptosystem [24]. They provide a practical security evaluation but no formal proof. Recently, attacks on PPE have been published [16, 23]. Alternative systems for encrypted databases refraining from PPE leave server-side secure aggregation as open problem [22].

*Seabed* combines additively symmetric homomorphic encryption (ASHE) and deterministic encryption to support aggregation queries with grouping [25]. ASHE has improved performance in comparison to asymmetric schemes such as Paillier, and is optimized for dense datasets, where multiple consecutive values are aggregated. This optimization, however, has issues in combination with additional filtering, e.g., expressed by `WHERE` clauses. In the worst case, the client's computation overhead for GroupBy statements in combination with `WHERE` clauses is the same as decryption and aggregation of all records matching the `WHERE` clause. Similar to the construction presented in Section 3.1, the authors encode the value attribute and the group attribute into a single plaintext and encrypt it using ASHE, hence they avoid deterministic encryption. Instead of using ciphertext packing, they create a new column for each group attribute value. To limit storage consumption, they propose an enhanced scheme that only introduces new columns for common group attribute values and one single column for less frequent ones. A column with deterministically encrypted group values is added to be able to aggregate the column of uncommon values correctly. This column only contains values in rows with uncommon group values, thus rows with common group values can be used to flatten the histogram by adding dummy values with no impact on the aggregation result. Seabed does not support grouping operations with multiple attributes natively.

In Table 11 we compare all schemes reviewed in this section regarding their support for aggregation and grouping of multiple attributes directly on the server, we classify their security properties and if these are proven formally.

## 8 CONCLUSION

We propose *SAGMA*, an encryption scheme for secure aggregation grouped by multiple attributes using ciphertext packing and dynamic shifting using somewhat homomorphic encryption. Our scheme is compatible with additional filtering clauses in the aggregation query and supports database updates. Previous constructions either leak more information or do not support grouping by multiple attributes together with additional filter clauses.

We partition the grouping attributes into disjoint buckets where values mapped into the same bucket remain indistinguishable during query execution. The choice of parameters, such as the bucket size or the assignment algorithm for group values to buckets, results in different trade-offs between security, supported queries, computational overhead and storage requirements. We analyze the theoretical implications of these parameters and evaluate combinations of concrete instances in a prototypical implementation.

# REFERENCES


[1] A. Akavia, D. Feldman, and H. Shaul. Secure search on encrypted data via multi-ring sketch. In *Proceedings of the 25th ACM Conference on Computer and Communications Security*, CCS, 2018.

[2] E. B. Barker. Recommendation for key management, part 1: General. Technical report, National Institute of Standards & Technology, Gaithersburg, MD, United States, 2016.

[3] J. Bater, G. Elliott, C. Eggen, S. Goel, A. N. Kho, and J. Rogers. SMCQL: secure query processing for private data networks. *PVLDB*, 10(6), 2017.

[4] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-DNF formulas on ciphertexts. In *Theory of Cryptography Conference*, TCC, 2005.

[5] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx. Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution. In *Proceedings of the 27th USENIX Security Symposium*, USENIX, 2018.

[6] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology*, Crypto. 2013.

[7] A. Ceselli, E. Damiani, S. D. C. D. Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Modeling and assessing inference exposure in encrypted databases. *ACM Transactions on Information and System Security (TISSEC)*, 8(1):119–152, 2005.

[8] V. Costan and S. Devadas. Intel sgx explained. *IACR Cryptology ePrint Archive*, 2016:86, 2016.

[9] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS, 2006.

[10] M. Ermolov and M. Goryachy. How to hack a turned-off computer, or running unsigned code in intel management engine. In *Black Hat Europe*, 2017.

[11] S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M. Rosu, and M. Steiner. Rich queries on encrypted data: Beyond exact matches. In *European Symposium on Research in Computer Security*, pages 123–145, 2015.

[12] D. M. Freeman. Converting pairing-based cryptosystems from composite-order groups to prime-order groups. In *Advances in Cryptology – EUROCRYPT 2010*, pages 44–61. Springer Berlin Heidelberg, 2010.

[13] B. Fuller, M. Varia, A. Yerukhimovich, E. Shen, A. Hamlin, V. Gadepally, R. Shay, J. D. Mitchell, and R. K. Cunningham. Sok: Cryptographically protected database search. *arXiv preprint 1703.02014*, 2017.

[14] T. Ge and S. Zdonik. Answering aggregation queries in a secure system model. In *Proceedings of the 33rd international conference on Very Large Data Bases*, VLDB, 2007.

[15] C. Gentry et al. Fully homomorphic encryption using ideal lattices. In *STOC*, volume 9, pages 169–178, 2009.

[16] P. Grubbs, K. Sekniqi, V. Bindschaedler, M. Naveed, and T. Ristenpart. Leakage-abuse attacks against order-revealing encryption. Cryptology ePrint Archive, Report 2016/895, 2016.

[17] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, SIGMOD, 2002.

[18] F. Hahn, N. Loza, and F. Kerschbaum. Practical and secure substring search. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, SIGMOD, 2018.

[19] B. Hore, S. Mehrotra, M. Canim, and M. Kantarcioglu. Secure multidimensional range queries over outsourced data. *The VLDB Journal—The International Journal on Very Large Data Bases*, (3):333–358, 2012.

[20] B. Hore, S. Mehrotra, and G. Tsudik. A privacy-preserving index for range queries. In *Proceedings of the 30th international conference on Very Large Data Bases*, VLDB, 2004.

[21] Y. Hu, W. J. Martin, and B. Sunar. Enhanced flexibility for homomorphic encryption schemes via CRT. In *Applied Cryptography and Network Security (ACNS)*, 2012.

[22] S. Kamara and T. Moataz. Sql on structurally-encrypted databases. In *International Conference on the Theory and Application of Cryptology and Information Security*, ASIACRYPT. Springer, 2018.

[23] M. Naveed, S. Kamara, and C. V. Wright. Inference attacks on property-preserving encrypted databases. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security*, CCS, 2015.

[24] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Eurocrypt*, 1999.

[25] A. Papadimitriou, R. Bhagwan, N. Chandran, and R. Ramjee. Big data analytics over encrypted datasets with seabed. In *12th USENIX Symposium on Operating Systems Design and Implementation*, OSDI, 2016.

[26] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*, SOSP, 2011.

[27] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica. Opaque: An oblivious and encrypted distributed analytics platform. In *NSDI*, pages 283–298, 2017.