Joins over Encrypted Data with Fine Granular Security

Florian Hahn
SAP SE
Karlsruhe, Germany
florian.hahn@sap.com

Nicolas Loza
SAP SE
Karlsruhe, Germany
nico.loza@gmail.com

Florian Kerschbaum
University of Waterloo
Waterloo, Canada
fkerschb@uwaterloo.ca

Abstract

Performing joins over encrypted data is particularly challenging, since the query result or access pattern of $1:n$-joins reveals the frequency of each distinct element in the column. This frequency information is used in many easy, but very detrimental inference attacks. In this paper we present a different approach: Instead of implementing a stand-alone join operator that reveals the frequency of each element in the column, we show how to construct joins over encrypted data after selection operations have been applied. These joins only leak the fine granular access pattern and frequency of elements selected for the join. Our new fine-granularly secure joins use searchable encryption and key-policy attribute-based encryption and support dynamically adding and removing database rows. Their performance is practical and we present an implementation in MySQL.

I. Introduction

When performing database operations over encrypted data leakage from the ciphertexts and access patterns enables sometimes detrimental attacks [25], [30], [43]. In particular, implementing secure join operations seems to be challenging. Consider a common $1:n$ equi-join: Each of the n elements where n is different for each distinct element is joined to one element of the other column. Just the result of this join operation – or the access pattern that created it – reveals the frequency, i.e. n, of each distinct element. Observing a single of these operations enables the frequency analysis attacks described in [25], [30], [43]. Padding all distinct elements to the same frequency is very space- and time-consuming, since their distribution may be uneven, and is difficult to maintain under updates.

All previous works securing join operations either require secure hardware [5], [3], [40] or ignore the access pattern [48], [13], [46]. Secure hardware shifts the trust to the hardware manufacturer which has been put into question given recent attacks [41], [53]. Mechanisms relying only on cryptography implemented in software [48], [13], [46] follow this model: For each pair of columns to be joined there is some secret information, e.g. a secret key. The client reveals this information only when a join is performed between those columns by the server. Before the secret key is revealed, the ciphertexts can be semantically secure. However, after the secret key is revealed, the frequency of each distinct element is revealed. We call this an “all-or-nothing disclosure”.

In this paper we present a “fine-granular disclosure” for join operations. We leverage the following observation for security: Each query optimizer pushes selection operations down towards the bottom of the query tree in order to minimize the number of tuples to perform joins over. As a consequence, we only need to reveal the frequency of elements selected for a join. This requires that there is no longer a single secret key, but that each selected element reveals secret information for its join. However, as we show in Section II-C naively combining existing methods for securing joins, such as [48], and fine-granular encryption for each element does not result in optimal security. Instead, we use key-policy attribute-based encryption and searchable encryption in Section III in order to construct an improved scheme that further reduces the leakage. We give a formal upper bound on the leakage of our scheme in Section III-C. The performance of our fine-granular joins remains practical and we present an evaluation of an implementation in MySQL in Section IV. Furthermore, our scheme is dynamic allowing for database rows to be added and deleted with little overhead.

Our contributions can be summarized as follows

- We present the first scheme that does not offer all-or-nothing security, but protects sensitive data on a finer granularity while preserving controlled equi-join functionality.
- We quantify the information leaked by the execution of our secure join protocol and prove this leakage as an upper bound.
- We show that our scheme is dynamic, that is, the client can insert new data after the initial database encryption with guaranteed security properties.
- We implemented and evaluated our proposed solution for secure joins on a real system based on MySQL and Java.

The remainder of the paper is structured as follows: In Section II we motivate the abstract problem we address in this paper, illustrate it with a small example and define a formal framework addressing this problem. Further, we give a strawman construction that implements that framework and discuss several drawbacks of this construction. In Section III we review cryptographic tools required for the solution addressing the drawbacks discussed in the previous section, quantify the information-leakage of this construction and give a simulation proof. Based on a prototypical implementation, we report performance numbers for the construction in Section IV. Finally, we revise the related work for the domain of encrypted databases in Section V before we conclude our work in Section VI.

II. Problem Description

In order to decrease data redundancy and increase data consistency for data stored in relational databases, the process of database normalization introduced by Codd [17] is applied during the database design phase. We support joins for tables in third normal form, that is, all tables contain only columns that are non-transitively dependent on the primary key. This is achieved by splitting the table into two tables, where previously depended data is stored in its own separate table. The dependency is then modeled as primary key in the one table, and foreign key in the second table and can be reconstructed using the join operation in the data query, e.g. a SQL `select` statement.

Assuming two tables $\mathbf{T_0}, \mathbf{T_1}$, the result of the *equi-join* operation on two join columns, one from $\mathbf{T_0}$ and one from $\mathbf{T_1}$ is the set of all combinations of rows from $\mathbf{T_0}, \mathbf{T_1}$ that contain equal values in their join columns. More formally, table $\mathbf{T_0}$ has schema $(PK_{\mathbf{T_0}}, A_1, \ldots, A_l)$ with primary key $PK_{\mathbf{T_0}}$ and attributes $A_1, \ldots, A_l$, this table consists of $|\mathbf{T_0}|$ records $(pk^1_{\mathbf{T_0}}, a^1_1, \ldots, a^1_l), \ldots, (pk^{|\mathbf{T_0}|}_{\mathbf{T_0}}, a^{|\mathbf{T_0}|}_1, \ldots, a^{|\mathbf{T_0}|}_l)$; table $\mathbf{T_1}$ has schema $(FK_{\mathbf{T_0}}, B_1, \ldots, B_m)$ with foreign key $FK_{\mathbf{T_0}}$ establishing the relationship to table $\mathbf{T_0}$ and attributes $B_1 \ldots, B_m$, this table consists of $|\mathbf{T_1}|$ records $(fk^1_{\mathbf{T_0}}, b^1_1, \ldots, b^1_m), \ldots, (fk^{|\mathbf{T_1}|}_{\mathbf{T_0}}, b^{|\mathbf{T_1}|}_1, \ldots, b^{|\mathbf{T_1}|}_m)$. In the following we use the row number as row ID, e.g. the the third row in $\mathbf{T_0}$ has row ID 3 with primary key pk^3.

The equi-join with *join attributes* $PK_{\mathbf{T_0}}$ and $FK_{\mathbf{T_0}}$ is an operation with table $\mathbf{T_0}$ and table $\mathbf{T_1}$ as input and denoted as $\mathbf{T_0} \bowtie \mathbf{T_1}$. The result of $\mathbf{T_0} \bowtie \mathbf{T_1}$ has schema $(PK_{\mathbf{T_0}}, A_1, \ldots, A_l, B_1, \ldots, B_m)$ and consists of all records $(pk^i_{\mathbf{T_0}}, a^i_1, \ldots, a^i_l, b^j_1, \ldots, b^j_m)$ with matching keys $pk^i_{\mathbf{T_0}} = fk^j_{\mathbf{T_0}}$ for all $i \in [1, |\mathbf{T_0}|], j \in [1, |\mathbf{T_1}|]$. Note, that the primary keys $pk^i_{\mathbf{T_0}}$ in table $\mathbf{T_0}$ need to be unique, but each primary key maps to possible multiple foreign keys $fk^j_{\mathbf{T_0}}$. We assume further filtering based on additional filtering-predicates chosen from $\{A_1, \ldots, A_l\}$ and $\{B_1, \ldots, B_m\}$.

For example, consider the relations *Employee* and *Dept* as defined in Table I with foreign key "Dept" and Table II with primary key "DName". In SQL one instance for an equi-join query we aim to support on encrypted databases with minimal information-leakage is

$$\texttt{SELECT * FROM } Emp \texttt{ JOIN } Dept \texttt{ ON } Dpt = DName$$
$$\texttt{WHERE } Mngr = "Eve" \texttt{ AND } Name = "Sally"$$

with the corresponding result table shown in Table III that is inevitably leaked to the service provider.

We emphasize that our scheme is not restricted to joins between primary and foreign keys in normal form. Any join

Name	EmpId	Dept
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Eve	2202	Sales
Sally	3737	Production
Sally	1245	Production

TABLE I: *Emp*

DName	Mngr
Finance	George
Sales	Eve
Marketing	Eve
Production	Charles

TABLE II: *Dpt*

Name	EmpId	DName	Manager
Sally	2241	Sales	Eve

TABLE III: *Emp* $\bowtie$ *Dpt* `WHERE` Mngr="Eve" and Name="Sally"

operation between any set of columns can be implemented by the database designer in the same manner as those from normal form. However, joins between primary and foreign keys are a prime example where the database designer must foresee join operations and we hence use it as an example.

A. Framework

The framework of our secure-join scheme with support of the filtering predicates described in the previous section can be expressed as follows:

Definition 1 (Secure-Join Scheme): Let $\mathbf{T_0}$ and $\mathbf{T_1}$ be the tables to be encrypted and joined. A scheme SecJoin supporting *secure joins with value blinding* must implement the following algorithms:

- $MK \leftarrow$ Setup(λ). The setup algorithm takes as input a security parameter λ. It outputs the master key MK.
- $c \leftarrow$ EncRow$(MK, i, jv, \mathbf{s})$. The encrypt row algorithm takes as input the master key MK, the indicator i indicating the type of join value jv (i.e. jv is a foreign key if $i = 1$, primary key otherwise) and the corresponding row attributes $\mathbf{s}$. It outputs an encrypted join value c that is compatible with table $\mathbf{T}_i$ and can be joined with table $\mathbf{T}_{1-i}$.
- $\mathbf{C}_i \leftarrow$ EncTab$(MK, \mathbf{T}_i)$. The encrypt table algorithm takes as input a table $\mathbf{T}_i$ and the master key MK. It runs EncRow for every row in $\mathbf{T}_i$ and returns the collection of all resulting encrypted join values in form of an encrypted table C_i.
- $\mathbf{C} = (\mathbf{C_0}, \mathbf{C_1}) \leftarrow$ EncDB$(MK, \mathcal{D})$. This algorithm encrypts each table in database $\mathcal{D} = (\mathbf{T_0}, \mathbf{T_1})$ using EncTab and returns the resulting encrypted database.
- $\tau_{jq} \leftarrow$ GenToken(MK, jq). The token generation algorithm takes the master key MK and a *join query* jq consisting of additional conditions on the attribute predicates for the tables in $\mathcal{D}$, e.g. specified via a `WHERE` clause in SQL. It returns a join token τ_{jq} for the corresponding query.
- $M_{\mathbf{T_0} \to \mathbf{T_1}} \leftarrow$ Join$(\mathbf{C_0}, \mathbf{C_1}, \tau_{jq})$. The join algorithm takes as input the two encrypted tables $\mathbf{C_0}$ and $\mathbf{C_1}$, together with join token τ_{jq}. The result is a map $M_{\mathbf{T_0} \to \mathbf{T_1}}$, which maps row IDs in $\mathbf{T_0}$ to their sets of matching row IDs in $\mathbf{T_1}$.

Note that this scheme can be extended to support joins over multiple tables with the same foreign key column to be joined on, i.e., for all tables $\mathbf{T}_j$ that contain the foreign key call EncRow with indicator $i = 1$.

B. Security

Utilizing the simulation-based security proof as introduced for Symmetric Searchable Encryption by Curtmola et al. [18] we quantify the leakage of secure-join schemes. Note that this leakage quantification is an upper bound – in the following we give an intuition of the security proof for such leakage bounds. From a technical point of view we construct a simulator $\mathcal{S}$ for all algorithms that require sensitive data as input, however, the simulator cannot access the actual sensitive data but only the result of leakage function $\mathcal{L}$ of that data. Given such a simulator $\mathcal{S}$ we prove that any PPT adversary $\mathcal{A}$ is not able to distinguish whether he has been provided with true encryption values and join-tokens or simulations thereof. Following the framework given in Definition 1, $\mathcal{S}$ simulates encryption algorithm EncRow and the algorithm for join-tokens generation GenToken. Note that the other operations with which the client can encrypt data, EncTab and EncDB, are based upon EncRow itself. Assuming there exists an adversary $\mathcal{A}$ that can successfully distinguish between true encryption values and their simulated version, this implies that the adversary has additional leakage that was not given to the simulator, which is a contradiction to the constructed simulator $\mathcal{S}$.

A more formal description of this idea is given in the following definition:

Definition 2: Let SecJoin = (Setup, EncRow, EncTab, EncDB, GenToken, Join) be a secure-join scheme, $\lambda \in \mathbb{N}$ be the security parameter. Consider the following probabilistic experiments with a stateful PPT attacker $\mathcal{A}$, a stateful simulator $\mathcal{S}$, and a stateful leakage function $\mathcal{L}$:

- $\mathbf{Real}_{\mathcal{A}}^{\text{SecJoin}}(\lambda)$: the challenger runs Setup(λ) to get the master key MK. The adversary $\mathcal{A}$ generates a polynomial set of non-adaptive encryption requests $r_1, ..., r_q$ with $r_j = (i, jv, \mathrm{s})$ and $i \in \{0, 1\}$, with $q = poly(\lambda)$. Along with these, $\mathcal{A}$ also generates a polynomial set of non-adaptive join queries $jq_1, ..., jq_{\widehat{q}}$, with $\widehat{q} = poly(\lambda)$. $\mathcal{A}$ then sends $(\{r_1, ..., r_q\}, \{jq_1, ..., jq_{\widehat{q}}\})$ to the challenger. For each encryption request, the challenger generates a ciphertext
$$c \leftarrow \text{EncRow}(MK, i, jv, \mathrm{s}),$$
and for each join query, the challenger generates
$$\tau_{jq} \leftarrow \text{GenToken}(MK, jq).$$
The challenger then returns all ciphertexts $c_1, ..., c_q$ and all join tokens $\tau_1, ..., \tau_{\widehat{q}}$ to $\mathcal{A}$. Finally, $\mathcal{A}$ returns a bit b that is the output of the experiment.
- $\mathbf{Sim}_{\mathcal{A},\mathcal{S}}^{\text{SecJoin}}(\lambda)$: the simulator sets up its internal environment. The adversary $\mathcal{A}$ generates a polynomial number of non-adaptive encryption requests, as well as a polynomial number of non-adaptive join queries, and sends them all to $\mathcal{S}$. For this complete non-adaptive query set, the simulator $\mathcal{S}$ is provided with the corresponding leakage $\mathcal{L}$. Using this leakage, $\mathcal{S}$ simulates and returns the appropriate ciphertexts $\widetilde{c}$ or join tokens $\widetilde{\tau_{jq}}$. Finally, $\mathcal{A}$ returns a bit b that is the output of the experiment.

We define the advantage of $\mathcal{A}$ as
$$\left|\Pr\left[\mathbf{Real}_{\mathcal{A}}^{\text{SecJoin}}(\lambda) = 1\right] - \Pr\left[\mathbf{Sim}_{\mathcal{A},\mathcal{S}}^{\text{SecJoin}}(\lambda) = 1\right]\right|.$$
We say SecJoin is semantically $\mathcal{L}$-secure if for all polynomial-sized (in λ) non-adaptive adversaries $\mathcal{A}$ there exists a non-uniform polynomial-sized simulator $\mathcal{S}$, so that the advantage of $\mathcal{A}$ is negligible in λ.

In order to benefit from all further progress achieved in both active research topics, attribute-based encryption and searchable symmetric encryption, we use these cryptographic tools as black boxes in our security proof. As a result, the information leakage we define for the security proof may be too pessimistic for some possible tools in the sense that the quantified leakage is an over-estimation. Further, we aim for security against non-adaptive attackers due to our black box applications of attribute based encryption. While it is easy to build efficient SSE systems that are secure even against adaptive attackers, the construction of ABE schemes that are secure against adaptive attackers is really challenging as discussed by Boneh et al. [11]. However, this flexibility in the choice of actual implementations of attribute-based encryption and searchable symmetric encryption results in several possible trade-offs between performance and information leakage. Depending on the offered properties of the tools actually implemented in our construction, the information leakage might be lower than the leakage we state in Section III-C.

C. Straw-man solution

In this subsection we provide a straw-man implementation of such framework that increases the security compared to all current solutions [48], [13], [46], [54] for databases with *only one* additional attribute column. That is, our construction does not provide the previously mentioned all-or-nothing security, but rather information-leakage on a finer granularity. In this case, $\mathbf{T_0}$ has schema (PK_A, A_1) and $\mathbf{T_1}$ has Schema (FK_A, B_1) with join-attributes PK_A, FK_A respectively, and a filter clause jq with exactly one filtering predicate a for attribute A_1 and one filtering predicate b for B_1. For this straw-man construction we follow the idea by Popa et. al [48] and utilize deterministic encryption for protecting the join values. For our example Table I, a solution solely based on deterministic encryption would reveal the frequency of all 3 distinct values stored in the "Dept" column. Thus, in order to minimize the leakage of a secure-join scheme SecJoin, we observe that it is sufficient to unveil the join result for all rows that match the additional filtering clause jq, i.e. having value a for column A_1 in $\mathbf{T_0}$ and value b for column B_1 in $\mathbf{T_1}$. Thus, instead of joining the complete tables $\mathbf{T_0}$ and $\mathbf{T_1}$ followed by a filtering on that join-result, we follow the orthogonal approach and run a privacy-preserving filtering on the complete tables $\mathbf{T_0}$ and $\mathbf{T_1}$, and perform the equi-join operation on that filtered result set. This is enforced by additional encapsulation of all join values using a semantically secure encryption scheme (Enc, Dec) keyed with a secret key k_a and k_b derived from the attribute predicates that are used for additional filtering. Further, we assume that the decryption algorithm Dec indicates successful decryption. That is, decryption $\text{Dec}(k', \text{Enc}(k, m))$ is called successful iff. $k' = k$; this can be implemented by, e.g., concatenating the hash value $h(m)$ to the encryption: $\text{Enc}(k, m||h(m))$ and checking this relation in the decryption algorithm. Given a deterministic encryption scheme $\Pi^{\text{Det}} = (\text{Gen}^{\text{Det}}, \text{Enc}^{\text{Det}}, \text{Dec}^{\text{Det}})$, a semantically secure encryption scheme $\Pi = (\text{Enc}, \text{Dec})$ with keyspace $\mathcal{K}$ and a key derivation function $\text{KDF} : \{0,1\}^\lambda \times \{0,1\} \times \{0,1\}^* \to \mathcal{K}$, we can implement a secure-join scheme supporting one attribute per table according to the framework given in Definition 1 as follows:

- $MK \leftarrow \text{Setup}(\lambda)$. Output $MK \leftarrow \text{Gen}^{\text{Det}}(1^\lambda)$.
- $c \leftarrow \text{EncRow}(MK, i, jv, s)$. Sample $SK_s \leftarrow \text{KDF}(MK, i, s)$, encrypt jv under key SK_s as $\text{Enc}(SK_s, \text{Enc}^{\text{Det}}(MK, jv))$, and output the resulting ciphertext.
- $\mathbf{C}_i \leftarrow \text{EncTab}(MK, \mathbf{T}_i)$. For every row in $\mathbf{T}_i$ call EncRow and return $\mathbf{C}_i = \{\text{EncRow}(MK, i, k^j, s^j)\}_{j \in [1, |\mathbf{T}_i|]}$.
- $C = (\mathbf{C_0}, \mathbf{C_1}) \leftarrow \text{EncDB}(MK, \mathcal{D})$. This algorithm encrypts both tables in database $\mathcal{D} = (\mathbf{T_0}, \mathbf{T_1})$ using EncTab and returns the resulting ciphertext tables.
- $\tau_{jq} \leftarrow \text{GenToken}(MK, jq)$. Note, that in this construction $jq = (a, b)$ consists of exactly two attributes, a for attribute A_1 and b for attribute B_1. Derivate the corresponding encapsulation keys $SK_0 \leftarrow \text{KDF}(MK, 0, a)$, $SK_1 \leftarrow \text{KDF}(MK, 1, b)$ and return join token $\tau_{jq} = (SK_0, SK_1)$ for the corresponding query.
- $M_{\mathbf{T_0} \to \mathbf{T_1}} \leftarrow \text{Join}(\mathbf{C_0}, \mathbf{C_1}, \tau_{jq})$. Parse token $\tau_{jq} = (SK_0, SK_1)$. For every encrypted join value $ej^j \in C_i$ use SK_i for decrypting $\text{Enc}^{\text{Det}}(MK, jv) = \text{Dec}(SK_i, ej^j)$. For $pd^i = \text{Enc}^{\text{Det}}(MK, jv)$ in $\mathbf{C_0}$ that is decrypted successfully, create map entry $M[i]_{\mathbf{T_0} \to \mathbf{T_1}} \leftarrow \{j : fd^j = \text{Enc}^{\text{Det}}(Mk, jv) \in \mathbf{C_1} \text{ with } fd^j = pd^i\}$. Finally, output the complete mapping $M_{\mathbf{T_0} \to \mathbf{T_1}}$.

In the following, we discuss three drawbacks of this straw-man construction and give formal comprehensive solutions in the next section.

First, orthogonal to security, performance is bad in this specific solution: filtering is linear in the table size, that is, all rows must be decrypted and checked for a successful decryption. This search overhead can be reduced by the application of a dynamic and efficient searchable symmetric encryption scheme based on inverted indexing. We emphasize that we assume the join values as sensitive data, hence we strive to minimize the leakage that can be extracted in regards to the secure-join operations and the underlying join values. However, we do not address the information leakage for the row attributes and the additional leakage induced by such inverted indexing techniques. This leakage has been studied in various papers about dynamic and efficient searchable symmetric encryption, such as [32], [52], [44]. Hence, we assume the secure application of such dynamic and efficient searchable symmetric encryption schemes resulting in more efficient pre-filtering step on the predicates. In the following discussion about secure joins we solely focus on protecting the join values (i.e. both primary and foreign key) but do not consider the security of additional attribute predicates, nor do we handle the potential pre-filtering.

Second, the application of deterministic encryption for securing the join values has additional leakage that is no direct consequence of the join result. That is, all values that match the `WHERE` clause in table $\mathbf{T_1}$ leak that they have the same foreign key fk even if they are not part of the equi-join result, e.g. in our example Table I, the rows matching the Name="Sally" are decrypted successfully unnecessarily unveiling the frequency of the foreign key "Production". As observed by Pang et al. [46], this enables the server to extract the result of a self-join, although not queried explicitly by the client.

Third, the required memory of the straw-man construction grows exponentially in the number of attributes for each table. In more detail, given table $\mathbf{T_0}$ with schema $(PK, A_1, \ldots A_n)$ and the possibility to filter for all n attributes, the protected join value must be blinded with all possible combinations of the n attributes resulting in 2^n different keys SK_i^j for $j \in [1, 2^n]$ and the resulting blinded encrypted join values for all 2^n different keys must be stored for each row in $\mathbf{T_0}$. The analogue argument holds true for table $\mathbf{T_1}$.

While the first drawback is addressed extensively by previous work, we focus on the latter two for the rest of our work.

III. Implementation

For the identified problems of the straw-man construction we identified different special cryptographic tools required to address these problems and achieve our goal to reduce the information-leakage induced by join operation on a finer granularity. We revise these required tools in the following Subsection III-A. Putting these tools together we present a comprehensive description of our implementation fitting the framework for secure-joins as specified in Definition 1. We quantify an upper bound for the information-leakage induced by our implementation based on the security properties offered by the used tools, and prove this upper bound in the formal framework provided in Definition 2.

A. Required Tools

Recall that we assume two different tables $\mathbf{T_0}, \mathbf{T_1}$ where the join values of $\mathbf{T_0}$ are primary keys, hence they are unique, while the join values of $\mathbf{T_1}$ are foreign keys, hence they might occur several times. As a result, the application of deterministic encryption Π^{Det} for equality checks on encrypted data as proposed in Subsection II-C has no consequences on the security level of encrypted values contained in join column of $\mathbf{T_0}$, however, weakens security for encrypted values contained in join column of $\mathbf{T_1}$. In order to minimize this security penalty while still providing the functionality of matching encrypted values for equality, we replace the deterministic encryption scheme with a searchable symmetric encryption scheme (SSE) as introduced by Song et al. [51], revised in the following.

Definition 3 (SSE Scheme): A secure Searchable Symmetric Encryption scheme SSE is a tuple of four (possibly probabilistic) polynomial-time algorithms:

- $K \leftarrow$ SSE-Setup(λ): is a probabilistic algorithm that takes as input a security parameter λ and outputs a secret key K.
- $c_w \leftarrow$ SSE-Enc(K, w): is a probabilistic algorithm that takes as input a secret key K and a plaintext w. It outputs a (randomized) encryption of w denoted as c_w.
- $t_w \leftarrow$ SSE-Token(K, w): is a deterministic algorithm that takes as input a secret key K and a plaintext w. It outputs a (deterministic) search token t_w.
- $r \leftarrow$ SSE-Match$(c_w, t_{w'})$: is a deterministic algorithm that takes as input a ciphertext c_w and a search token $t_{w'}$. It returns $r = 1$ if $w = w'$ with $c_w \leftarrow$ SSE-Enc(K, w) and $t_{w'} \leftarrow$ SSE-Token(K, w') using the same secret key K. Otherwise, this algorithm returns $r = 0$.

Note that the output of SSE-Enc is randomized even for the same input (i.e. the same key K and plaintext w), while the output of SSE-Token is deterministic. We denote both c_w and t_w as *SSE-values* in the following. In order to model the equi-join functionality for $\mathbf{T_0}$ and $\mathbf{T_1}$ using SSE, one encrypts all (unique) join-values of $\mathbf{T_0}$ calling SSE-Token and all (probably non-unique) join-values of $\mathbf{T_1}$ calling SSE-Enc. Here, the correct choice is crucial for the security, since the application of SSE-Enc on the non-unique values hides the frequency due to its randomized output characteristics making self-joins on $\mathbf{T_1}$ impossible.

We emphasize that, although $\mathbf{T_1}$ on its own is semantically secure after applying SSE-Enc, all values that occur in $\mathbf{T_0}$ as well have additional leakage due to the comparison functionality provided by SSE-Match. More formally, an honest-but-curious attacker can define sets

$$\{j \in [1, |\mathbf{T_1}|] : \exists i \in [1, |\mathbf{T_0}|] \text{ with } \\ \text{SSE-Match}(\text{SSE-Enc}(K, fk^j), \text{SSE-Token}(K, pk^i)) = 1\}$$

grouping randomized ciphertexts (or, in this case, their IDs) for the same underlying plaintext value. As a consequence, blinding the SSE-values (both SSE-ciphertexts and SSE-tokens) remains a vital security protection for the outsourced databases with support of secure-joins.

In order to solve the straw-man solution's exponential memory blowup, we introduce the concept of attribute-based encryption (ABE), which is an expansion of public key cryptography that allows the encryption and decryption of messages based on attributes assigned to the ciphertext during encryption time. Originally presented by Sahai et al. [50], it focused on ascribing the ciphertext with a predicate $f(\cdot)$, which then needed to be satisfied by the user's credential for the decryption to be successful. Later, Goyal et al. [24] defined this as *Ciphertext-Policy Attribute-based Encryption* (CP-ABE), while also defining its complementary: *Key-Policy Attribute-based Encryption* (KP-ABE). In the latter, attributes are used to annotate ciphertexts, and formulas over these are assigned to keys generated by the user. These formulas must then be satisfied by the attributes in the ciphertext for the decryption to be successful. For the purposes of this work, we will henceforth make use only of KP-ABE, and whenever we use the term ABE, we refer to KP-ABE.

In order to define the algorithms necessary for ABE, we first need to define what an access structure is.

Definition 4 (Access Structure): Let $P = \{P_1, ..., P_n\}$ be a set of parties. A collection $\mathbb{A} \subseteq 2^P$ is monotone if $\forall B, C : B \in \mathbb{A} \wedge B \subseteq C \rightarrow C \in \mathbb{A}$. An *access structure* (respectively, monotone access structure) is a collection (resp., monotone collection) $\mathbb{A}$ of non-empty subsets of P, i.e. $\mathbb{A} \subseteq 2^P \backslash \{\emptyset\}$. The sets in $\mathbb{A}$ are called the *authorized sets*, and the sets not in $\mathbb{A}$ are called the *unauthorized sets*.
Using this, we can now define the algorithms implemented by any KP-ABE scheme.

Definition 5 (KP-ABE Scheme): Given a message space $\mathcal{M}$ and access structure space $\mathcal{G}$, we define a Key-Policy Attribute-Based Encryption (KP-ABE) scheme as a tuple of the following (possibly probabilistic) polynomial-time algorithms:

- $(PK, MK) \leftarrow$ ABE-Setup(λ, U): is a probabilistic algorithm that takes as input a security parameter λ and an universe description U defining the set of allowed attributes in the system. It outputs the public parameters PK and the secret key MK.
- $CT \leftarrow$ ABE-Enc$(PK, M, \mathbf{S})$: is a probabilistic algorithm that takes as input the public parameters PK, a message M and a set of attributes $\mathbf{S}$, and outputs a (randomized) ciphertext CT associated with the attribute set.
- $SK_{\mathbb{A}} \leftarrow$ ABE-Key$(MK, \mathbb{A})$: is a probabilistic algorithm that takes as input the master secret key MK and an access structure $\mathbb{A}$, and outputs a (randomized) private key $SK_{\mathbb{A}}$ associated with the attributes described by $\mathbb{A}$.
- $M \leftarrow$ ABE-Dec$(SK_{\mathbb{A}}, CT)$: is a deterministic algorithm that takes as input a private key $SK_{\mathbb{A}}$ associated with access structure $\mathbb{A}$ and a ciphertext CT associated with attribute set $\mathbf{S}$ and outputs the message M encrypted in CT iff. $\mathbf{S}$ satisfies $\mathbb{A}$.

Note, that general KP-ABE schemes have no claims with respect to the security of the attribute set $\mathbf{S}$ used in ABE-Enc, hence one can potentially extract information about the used attribute set from a given ciphertext generated under this set. However, we assume the attribute set as sensitive relying on the stronger security property for the KP-ABE schemes called *attribute-hiding* as introduced by Katz et al. [34].

Further, in the standard definition of KP-ABE, it is specified using a finite attribute universe U. However, there are lines of work that propose implementations of KP-ABE with arbitrarily large attribute universes, like the one presented by Hohenberger and Waters [29]. Therefore, for future references, we will omit the usage of U. More importantly the access structures used to generate ABE-keys can be constructed from any boolean formula, as shown by Lewko and Waters [39], as well as from boolean formulas with threshold gates, shown by Liu et al. [42]. In our application, we will use ABE-keys to describe the restrictions placed upon a join query, e.g. the `WHERE` clause in a SQL query. It is therefore possible to support arbitrary restrictions described as boolean formulas, since ABE supports them as well. We emphasize that, following the approach of Kiayias et al. [38], this flexibility in the policy formulation can be utilized to allow range filtering with only logarithmic (in the value domain size) attribute blowup, e.g. a column stores values $v_i \in D$ that should be compatible with range queries in the `WHERE` clause in a SQL query.

For the sake of a brief and coherent security proof, however, our construction in the remainder of this work will focus only on *conjunctions*, i.e. formulas where all the specified restrictions must be fulfilled. Thus, from now on we will write ABE-Key$(MK, \{s_1, ..., s_l\})$, referring to the access structure describing the conjunction of all values $s_1, ..., s_l$.

It is worth noting that, depending on the construction of KP-ABE, the sizes of the ciphertexts produced by ABE-Enc and of the keys produced by ABE-Key can vary. For our intents and purposes, we will focus on performant constructions like [29], which requires only two pairings per decryption and none for encryption or key generation, and produces ciphertexts and keys whose sizes are linear in the number of attributes used for their generation.

B. Secure-Join Scheme

In summary, our straw-man construction is based on blinding deterministic encryption of the join values. The keys applied for the blinding are derived from their corresponding attribute predicates with the two drawbacks, namely i) an honest-but-curious attacker can deduce a self-join on $\mathbf{T_1}$ for all unblinded keys without being queried explicitly, and ii) the algorithm has an exponential memory-overhead in the number of attributes. Our implementation addresses both drawbacks, with the following approaches: i) the functionality of equality checks provided by deterministic encryption is realized with searchable symmetric encryption scheme $SSE =$ (SSE-Setup, SSE-Enc, SSE-Token, SSE-Match) (rf. Definition 3) thus rendering self-joins impossible, and ii) we reduce the memory-overhead to be linear in the number of attributes by using a key-policy attribute-based encryption scheme (rf. Definition 5) $ABE =$ (ABE-Setup, ABE-Enc, ABE-Key, ABE-Dec).

The general idea is visualized in Figure 1: At ①, the initial plaintext join values stored in both tables are protected using SSE, depending on indicator i either SSE-Token or SSE-Enc is called. At ②, the corresponding result is either a deterministic search token or a randomized ciphertext still enabling to calculate the complete equi-join of both tables. In order to further reduce the information leakage to the actual join-result we apply ABE to blind the SSE-values based on the filtering attributes for each table record at ③. The final ciphertext output by ABE-Enc is outsourced as depicted at ④. In order to delegate a join query the client calls ABE-Key transforming the involved filtering clause as attribute policy at ⑤. At ⑥, the ciphertexts encrypted under attributes matching the filtering

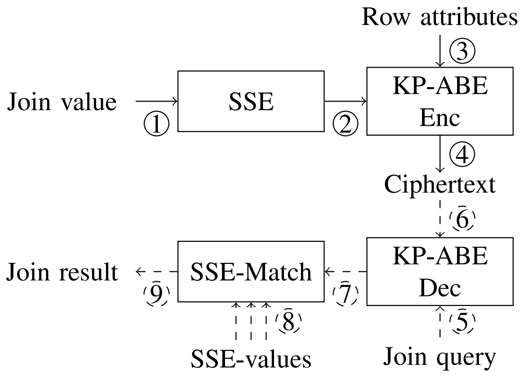

Fig. 1: Construction for joins with fine granular security. The solid arrows depict the encryption routine. The dashed arrows depict the join computation.

clause are successfully decrypted unveiling the underlying SSE value at ⑦. This unveiled value can be compared with all unveiled SSE-values contained in the table to be joined as sketched at ⑧. Note that steps ⑦ and ⑧ are repeated for each decrypted value. Finally, at ⑨ the complete values matching SSE-Match enable the server to retrieve the join result.

Formally, we implement secure-joins according to Definition 1 as follows:

- Setup(λ). Let $\lambda = (\lambda_0, \lambda_1)$ be the security parameter. The setup algorithm executes the following operations:
$$K_{\text{SSE}} \overset{\$}{\leftarrow} \text{SSE-Setup}(\lambda_0)$$
$$K_{\text{ABE0}} \overset{\$}{\leftarrow} \text{ABE-Setup}(\lambda_1)$$
$$K_{\text{ABE1}} \overset{\$}{\leftarrow} \text{ABE-Setup}(\lambda_1)$$
and returns $MK = (K_{\text{SSE}}, K_{\text{ABE0}}, K_{\text{ABE1}})$.
- EncRow$(MK, i, jv, \mathbf{s})$
$$sseVal \leftarrow \begin{cases} \text{SSE-Token}(K_{\text{SSE}}, jv) & \text{if } i = 0 \\ \text{SSE-Enc}(K_{\text{SSE}}, jv) & \text{if } i = 1 \end{cases}$$
$$c \overset{\$}{\leftarrow} \text{ABE-Enc}(abeKey_i, sseVal, \mathbf{s})$$
and returns c.
- EncTab$(MK, \mathbf{T}_i)$. Run EncRow for every row in $\mathbf{T}_i$, the SSE-values encapsulated by the ABE-encryption in form of an encrypted table $\mathbf{C}_i$.
- EncDB$(MK, \mathcal{D})$. Encrypt each table in a database $\mathcal{D} = (\mathbf{T_0}, \mathbf{T_1})$ using EncTab and return the resulting encrypted tables $C = (\mathbf{C_0}, \mathbf{C_1})$.
- GenToken(MK, jq). Let $jq = (jq_0, jq_1)$ be the attributes in the queries `WHERE` clause corresponding to columns in tables $\mathbf{T_0}$ and $\mathbf{T_1}$, respectively. This algorithm computes:
$$SK_{jq_0} \leftarrow \text{ABE-Key}(K_{\text{ABE0}}, jq_0)$$
$$SK_{jq_1} \leftarrow \text{ABE-Key}(K_{\text{ABE1}}, jq_1)$$
and returns $\tau_{jq} = (SK_{jq_0}, SK_{jq_1})$.
- $Join(\mathbf{C_0}, \mathbf{C_1}, \tau_{jq})$. Parse $\tau_{jq} = (SK_{jq_0}, SK_{jq_1})$ and remove the ABE blinding for all matching rows as

follows:

$$\begin{aligned}\mathbf{toks} = \{t^i \quad & \mid \exists\, encToken^i \in \mathbf{C_0} : \\ & t^i = \text{ABE-Dec}(SK_{jq_0}, encToken^i)\} \\ \mathbf{ciphs} = \{c^j \quad & \mid \exists\, encCiph^j \in \mathbf{C_1} : \\ & c^j = \text{ABE-Dec}(SK_{jq_1}, encCiph^j)\}\end{aligned}$$

It is important to note that both $\mathbf{toks}$ and $\mathbf{ciphs}$ are sets. While $\mathbf{toks}$ contains distinct values, $\mathbf{ciphs}$ may contain multiple SSE-ciphertexts for the same join-value. If either $\mathbf{toks}$ or $\mathbf{ciphs}$ is empty, then the equi-join result is empty, hence the algorithm returns $\perp$. Otherwise, define a map $M_{\mathbf{T_0} \to \mathbf{T_1}}$ where for every t_i search the matching ciphertexts, that is,

$$M[i]_{\mathbf{T_0} \to \mathbf{T_1}} \leftarrow \{j : c^j \in \mathbf{ciphs} \text{ with } \text{SSE-Match}(c^j, t^i) = 1\}$$

Finally, return $M_{\mathbf{T_0} \to \mathbf{T_1}}$.

Recall the assumption that table $\mathbf{T_0}$ has its primary keys as join values, hence we can assume that the join column contains only *unique values*. In this construction, we assume concretely that such values are the ones in $\mathbf{T_0}$'s join column, and thus we replace them with their corresponding SSE-tokens. The reasoning behind this is that SSE-ciphertexts are always randomized, and thus multiple encryptions of the same word cannot be recognized as such without a valid SSE-token. This is not the case with SSE-tokens, since they are generated deterministically. Further, all join-values that cannot be contained in the result set for the join query, due to not-matching the `WHERE` clause, remain obfuscated by attribute based encryption. In the next section we define this leakage intuition more formally and prove this leakage definition as an upper bound for our scheme.

C. Formal Security Properties

For the formal security analysis, let $\tau(H)$ be the trace induced by the SSE-values revealed through the join queries. Following the methodology of SSE published by Curtmola et al. [18], we define the trace of such q-query-history as

$$\tau(H) = \{|D_{id_1}|, \ldots, |D_{id_n}|, \alpha(H), \sigma(H)\},$$

where D_{id_j} is a document content protected by SSE, n is the number of such documents, $\alpha(H)$ is the access pattern matching queried keywords and $\sigma(H)$ is the search pattern identifying repeated keyword tokens. The access pattern $\alpha(H)$ consists of sets $\mathbf{D}(w_i) = \{id_j : \forall j \in [0, n] \text{ and } w_i \in D_{id_j}\}$ containing the documents matching the keyword w_i that has been searched for in query i. The search pattern $\sigma(H)$ indicates, whether two arbitrary searches were performed for the same keyword or not, i.e. whether the search token has been created for the same key word.

We re-interpret the scenario of SSE handling encrypted documents to our scenario handling exactly one encrypted join value per table row, resulting in $|D_{id_j}| = 1$ for all j. Furthermore, $\mathbf{D}(w_i)$ in $\alpha(H)$ will contain the identifiers of those rows whose join values match exactly the value w_i. Finally, the search pattern is empty, since we assume SSE-tokenization for unique primary keys, thus avoiding multiple occurrences of the same SSE-tokens.

In order to define the total leakage of our scheme, assume that a join query can be split as $jq = (\kappa_0, \kappa_1)$, where κ_0 and κ_1 represent the restrictions placed in jq regarding tables $\mathbf{T_0}$ and $\mathbf{T_1}$, respectively. Furthermore, for a set of restrictions $\kappa = \{a_1, ..., a_n\}$, we define the set $AP(\kappa) = \{ID(a_1), ..., ID(a_n)\}$, which we call attribute pattern, containing the identifiers of all restrictions placed in κ. Thus, if two join queries jq_i and jq_j have their restrictions on table $\mathbf{T_0}$ $\kappa_0^{jq_i}$ and $\kappa_0^{jq_j}$ such that there exists some $a \in \kappa_0^{jq_i} \cap \kappa_0^{jq_j}$, then $ID(a)$ will be present both in $AP(\kappa_0^{jq_i})$ and in $AP(\kappa_0^{jq_j})$.

Using definitions from above, let ID_0 and ID_1 be the information leaked through the join queries regarding tables $\mathbf{T_0}$ and $\mathbf{T_1}$, respectively. More specifically, for every id id_{jq} for join query jq, ID_0 maps said id to the tuple $(AP(\kappa_0^{jq}), \alpha_{jq})$, where $\alpha_{jq} = \alpha_1, ..., \alpha_l$ is the set of row IDs from table $\mathbf{T_0}$ matched by the restrictions in jq, i.e in κ_o^{jq}. ID_1 is similarly constructed, referring to columns and row IDs in table $\mathbf{T_1}$.

We sketch the proof of security as stated in Definition 2, with the following leakage function:

$$\mathcal{L}(r_1, ..., r_q, jq_1, ..., jq_{\widehat{q}}) = (\tau(H), ID_0, ID_1).$$

That is, we state a specific simulator with output computationally indistinguishable compared to the real protocol output. While the real protocol output is generated using real protocol input as well, the simulator has limit information modeled by $\mathcal{L}$.

In the following, we sketch the simulator and argue why this simulation is indistinguishable for any PPT adversary.

First, the simulator fills all rows that match (at least) one join-query with random values matching the characteristics leaked by the SSE leakage. This is not distinguishable by any attacker due to the SSE security. Next, $\mathcal{S}$ creates attribute predicates that are consistent with the join-queries and their attribute patterns extracted from the leakage function, i.e. the values are added to the correct columns and the corresponding rows share the same attributes. Further, ABE-keys are created for such fake predicates forming the simulated join-tokens. This is not distinguishable by any attacker due to the provided ABE security for ABE-keys. Finally, the simulator ABE-encrypts the two simulated tables with values that are consistent with the queries, as generated in the previous step, while also filling remaining empty attribute predicate cells with random values, i.e. all values that contain attributes matching no join-query. This is not distinguishable by any attacker due to the provided ABE security for ciphertexts. ■

IV. Performance Results

In this section, we provide an insight into the practical usage of our secure-joins scheme (see Section III-B). In order to do so, we implemented a prototype of our scheme, and tested its efficiency under three different aspects: 1) encryption, 2) query parsing, and 3) long-term performance.

Under the first setting, described in Section IV-A, we test the performance of the encryption step. This computation has to be executed on a trusted environment, converting the sensitive data to be outsourced into a cryptographically protected version preserving the functionality of joins. Under the second setting, described in Section IV-B, we measure the computational effort needed to transform a set of restrictions into an ABE-key. Recall, that this transformation is necessary in order to delegate the join computation without unveiling the complete join relation of the data encrypted and outsourced before. Our evaluation results allow us to assert that most modern personal (even mobile) devices are able to handle such operations in reasonable amounts of time, hence rendering join-token generation on the client-side realistic. Finally, under the third setting, described in Section IV-C, we measure the actual join execution time preformed in the untrusted environment. Initially completely obfuscated by the encryption step of our scheme, entries are gradually ABE-decrypted with every passing join query. The results of the latter test show that the performance impact of ABE lessens with time, and that queries with similar result sets tend to have decreasing cryptographic overhead, reducing the join computation time.

The following experiments were implemented in Java 8. All operations, i.e. client and server, were executed on one machine with 32 processors, each is a 64-bit Intel Xeon E5-2670 @2.60 GHz and with 240GiB RAM and running SUSE Linux Enterprise Server 11. In the operations involving a client and a server, the latter used a MySQL Server 5.7 instance for storage of the encrypted data. All tables were defined using the InnoDB storage engine. Moreover, our implementation makes use of the following libraries: Scapi for all "classical" cryptographic primitives (e.g. AES, SHA-X, HMAC, PRFs, etc), and jPBC for all group and pairing-based operations. Both are available as Java native code.

As SSE scheme we used a variation of the one proposed by Hahn and Kerschbaum [28], without the inverted index, where each "file" is a join value; and as PRF for the scheme we used a CBC-MAC-based PRF with AES as building block. As KP-ABE scheme we used that proposed by Hohenberger and Waters [29], with one of PBC's symmetric Type A pairings[1] over a group with a 160-bit-long prime number of elements, and a CBC-MAC-based PRF as the hash function necessary for the support of large universes. Moreover, in order to ascertain whether the decryption of an ABE-ciphertext was successful, we used MAC-then-encrypt with SHA-256 as MAC function.

A. Encryption

In order to test the efficiency and scalability of our SSE and ABE implementations, we generated random sets of rows, with different number of attribute columns, ranging from 3 to 20. Then, we proceeded to SSE-encrypt every row's join value, followed by ABE-encryption of the resulting ciphertext with the other row values as attributes. We chose to SSE-encrypt and not tokenize, since the encryption specified in [28] already contains the generation of a corresponding token. Thus, SSE-encryption *must* be slower than the tokenization. This way, we have a "worst-case" situation, although it must be noted that, unlike ABE, SSE-operations are all symmetric, and thus require minimal computational effort.

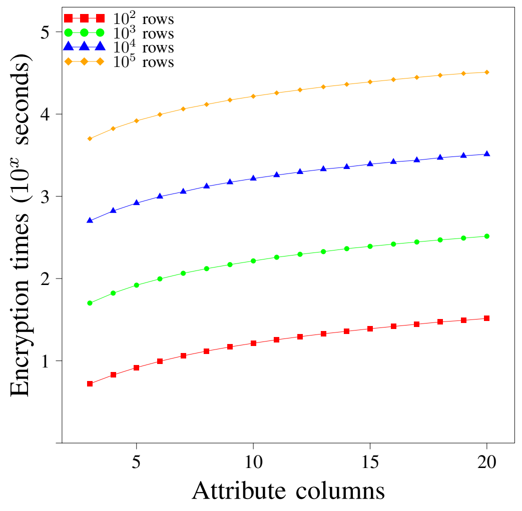

Fig. 2: Encryption times with varying rows and attribute columns

The results of executing these tests can be seen in Figure 2. It is evident that the performance of the encryption is linearly correlated with both the number of attribute columns ascribed to every ABE-ciphertext as well as with the number of rows to be encrypted. It is worth noting that these tests were executed purely and sequentially in Java, and only take into account the computational effort for the client to execute the necessary SSE and ABE operations, and do not include any transmission costs or I/O overhead, which would be observed when submitting the encrypted data to a (SQL) server. As such, it can be interpreted as the computational effort invested by a client into the encryption of a join column before outsourcing it. The complete process can also be easily parallelized.

B. Key Generation

Once the client has finished encrypting the database rows and outsourcing them, he can proceed to request from the server the data resulting from a join operation. In order to do so, the specified query's `WHERE` clause is parsed into two ABE-keys, which are sent to the server, who can use them to compute the requested join. In an independent test, with synthetically generated data, we measured the performance of generating a single ABE-key with varying number of restrictions (i.e. attributes), the results of which are presented in Figure 3. From these, we can gather that the key generation does not pose a serious challenge to any modern processor and can thus be successfully computed within a reasonable amount of time. Furthermore, this effort scales well, since it is linear in the number of conditions placed by the client in the `WHERE` clause.

[1]See https://crypto.stanford.edu/pbc/manual/ch08s03.html for specifics.

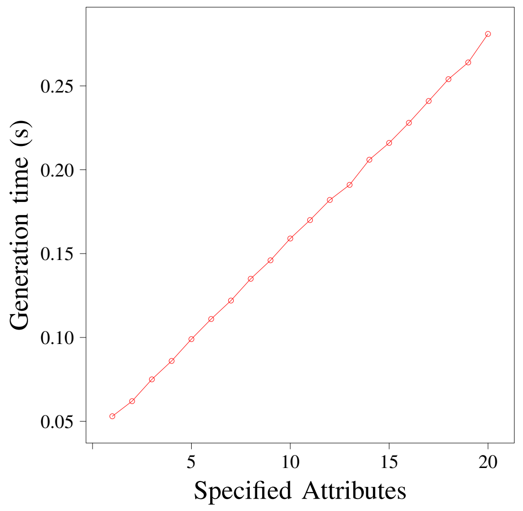

Fig. 3: ABE-key timings with varying attribute restrictions.

C. Trace Evaluation

For all experiments described in this section, we used data produced by the TPC Benchmark H[2]. Using this benchmark's data generator with a scaling factor of 0.1, we took the table $PART$ (20,000 rows and 6 attribute columns) with its primary key $P\_PARTKEY$ and the table $LINEITEM$ (600,000 rows and 8 attribute columns) with the foreign key $L\_PARTKEY$ as our test tables. After encrypting them with our Secure-Joins scheme (rf. Section IV-A) a random trace of join executions was generated. Iterating over said trace, the client parsed each join query into the corresponding join-token (rf. Section IV-B) and sent them to the server (running our protocol in Java, storing the databases in MySQL). Given this join-token, the secure join operation was computed and evaluated as discussed in the following.

In our tests, we assume that the server is able to quickly identify the rows satisfying the `WHERE` clause (e.g. through the usage of a Searchable Encryption scheme as highlighted in Section II-C), and thus proceeds to ABE-decrypt them, if necessary, and compare the underlying SSE-values. It is important to note that in this case, we took advantage of the 32 processors available in the test machine and parallelized internally each join query (not to be confused with parallel join queries). This was done in such a way that first, the (Java) server retrieved all rows from the (MySQL) tables $\mathbf{T_0}$ and $\mathbf{T_1}$ matching the corresponding restrictions placed in the `WHERE` clause. Once retrieved, all ABE-values were decrypted in parallel. Following, a "full join" was built between all matching rows of both tables, and the resulting set of rows was evenly distributed among the available threads. Each thread then proceeded to compute a "local" result set, which was then returned to the main thread once the computation was finished, so that all local results could be combined in a global result set. Finally, all values that needed ABE-decrypting were replaced

[2]Go to http://www.tpc.org/tpch/ for further information.

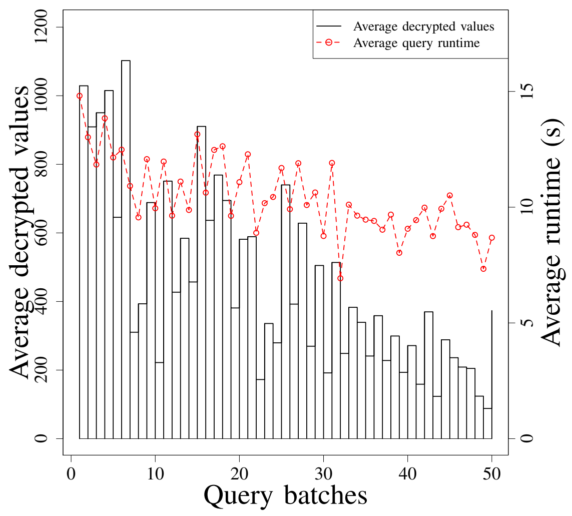

Fig. 4: Average decrypted values (stair steps) and average runtimes (dashed lines) for a trace with 10^3 joins queries, separated in batches of 20 queries.

with their underlying SSE-values in the corresponding MySQL tables, and the result set of the join operation returned to the client. The results of executing a trace with 10^3 join operations, with the server acting as specified before, can be seen in Figure 4. In it, for the sake of readability, we took the average runtime results from every 20 consecutive join queries, also referred to as "batch", and plotted them in red dashed lines, whereas the stairs steps represent the averaged number of ABE-values that needed to be decrypted per join batch. As we can see, in time (implicit in the x-axis) both plots tend to sink, since the queries will increasingly need to ABE-decrypt less values which, in turn, results in a lower average query runtime. Since the impact of ABE lessens in time, the dominating factor in later sets of queries (i.e. queries executed towards the end of the trace) is the number of necessary SSE-comparisons. This can be explicitly observed in Figure 5, which depicts (as before, averaged per batch) the ratios of the time invested by a single join query in executing ABE and SSE operations, compared to the operation's total runtime. As we can see, when starting the trace's execution, a join query invests close to 30% of its execution time on SSE-comparisons, and most of the rest is spent in ABE operations. This is contrasted with queries in later stages of the trace, where SSE comparisons take up more than 50% of the execution time, with some up to 60%. Since SSE-operations are much more efficient than ABE-operations, this results in lower runtimes for the queries executed later in the trace.

D. Impact of the Scaling Factor

Finally, we evaluate how our solutions scales with an increasing database size. Using the benchmark's data generator we created and encrypted databases of different sizes. Particularly, we performed the trace evaluation as described

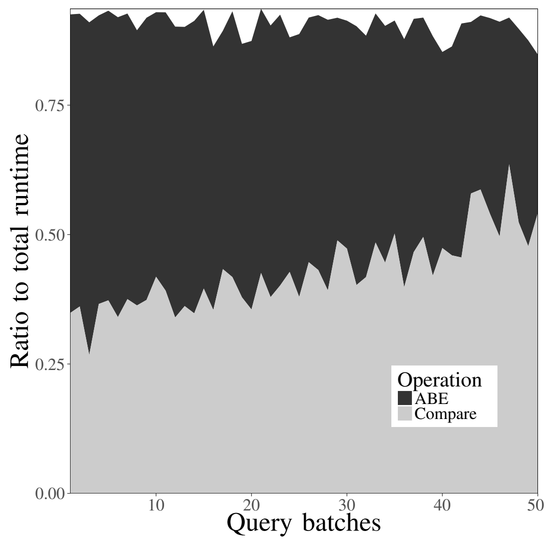

Fig. 5: Average ratios of time spent by a query in ABE and SSE-operations to the total runtime of the query.

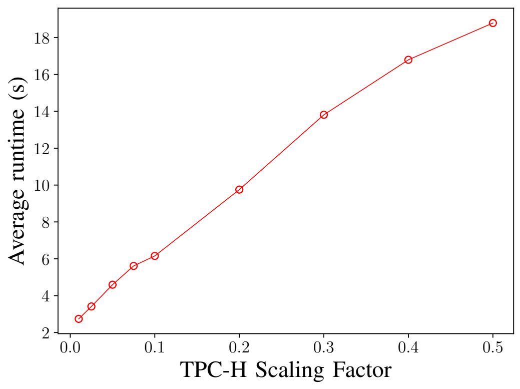

Fig. 6: Average join time for varying TPC-H scaling factors.

in previous Section IV-C for varying scaling factors ranging from 0.01 up to 0.5. As one can observe in Figure 6, the average runtime for one join operation increases linearly with this scaling factor. Here we calculated the average over 10^3 join operations.

The number of rows satisfying the `WHERE` clauses roughly increase linearly with the TPC-H scaling factor, the number of ABE-decryption calls and the result size increase linearly as well resulting in an overall linear increase.

V. Related Work

The security concerns in the database-as-a-service model has been addressed by multiple works previously [4], [26], [27]. The approach based on property-preserving encryption [7], [9], [45] studied extensively by Popa et al. [48] is adapted widely by commercial products [1], [2] due to its moderate integration effort and small computation overhead.

A. Secure Joins

Popa et al. [49] provide the functionality of secure joins by the utilization of deterministic encryption that offers the possibility of re-encryption [8]. That is, each column is encrypted deterministically with a different key, however, given a re-encryption token this key can be changed *without* an intermediate decryption. In the case of an equi-join query, the client creates such re-encryption token that allows to adjust the encryption key of one column matching the encryption key of the second table, enabling equi-joins by simple ciphertext comparison. For such adjustable joins Kerschbaum et al. [37] present strategies for optimizing the performance with respect to the required re-encryption. However, the security of deterministically encrypted data strongly depends on the underlying plaintext distribution and may be exploitable by an attacker as Naveed et al. demonstrated recently [43].

The idea of using secure coprocessors for computing joins on outsourced data has been introduced by Agrawal et al. [3]. Because of the low amount of memory available in these devices, the security of such methods relies on reading and writing from and to the processor in such a way, that the I/O access pattern leaks the smallest amount of information as possible. Agrawal et al. [3] then introduced an algorithm for such purposes. Li et al. [40] later showed that the security assumption from Agrawal et al. lead to unnecessary information leakage, and replaced it with a new definition, based on which they proposed three new algorithms for computing general joins on arbitrary predicates. Despite this, neither work [3], [40] provides a well-defined leakage formulation for their proposed algorithms. Arasu and Kaushik [5] give an approach for oblivious query processing that hides the access pattern of the join query and only unveils the join's result size. They call query processing oblivious iff. no attacker can distinguish the memory access sequence of two different databases and presented an algorithm fulfilling this security definition. However, all solutions based on secure hardware require additional trust assumptions regarding the hardware vendor which has been recently challenged [41], [53].

An alternative approach solely based on cryptographic assumptions is based on Secure Computation, specifically Private Set Intersection (PSI) as introduced by Fagin et. al [21] and formally studied by Freedman et al. [22]. Many protocols have been published, including ones theoretically most efficient [47] and practically deployed [55]. Outsourced PSI [35], [36] allows clients to upload their data to a server and then perform PSI which is closer to our system setting. However, all PSI protocols assume that sets contain only unique values. This constraint renders all these solutions unapplicable to secure joins due to possibly multiple occurrence of the same foreign key (i.e. join value) in one of the tables to be joined.

The first solution without this constraint has been proposed by Carbunar et al. [13] based on obfuscated Bloom filters constructed for the join values. For enabling the join operation, this obfuscation is removed, degenerating the security level for values stored in columns to be joined to that of deterministic

Scheme	Multiple foreign keys	Cryptographic assumptions only	Avoids Self-Joins	Quantified leakage
Deterministic Encryption [7], [48]	✓	✓	X	$N : M$
Trusted Hardware [3], [5], [40]	✓	X	✓	$n : m$
Private Set Intersection [35], [36]	X	✓	✓	$M : M$
Obfuscated Bloom Filters [13]	✓	✓	X	$n : m$
Pairing-based Join Schemes [46], [54]	✓	✓	✓	$N : M$
This paper	✓	✓	✓	$n : m$

TABLE IV: Comparison of related work. N and M is the total table size of the tables to be joined; n and m is the subset size resulting from additional filtering statements.

encryption. Further, due to the nature of Bloom filters, post-processing for removing false positives is required on the client side. In contrast, our solution does not raise false positives, hence client side computation is independent of the result set size. Pang et al. [46] propose a secret key encryption scheme and Wang et al. [54] propose a public key encryption scheme both rely on pairing based cryptography that allow equi-joins, encrypted by the client and by an untrusted third party, respectively. All schemes with support of non-unique join values published so far offer all-or-nothing security, in the sense that once the join has been performed the inner-join is unveiled completely. For a comparison of different approaches for secure database joins refer to Table IV.

a) Searchable Encryption: The idea of an encryption scheme providing functionality for exact pattern matching has been formulated by Song et al. [51]. Curtmola et al. [18] introduced the widely accepted simulation proof technique for static searchable symmetric encryption, further generalized for other applications by Chase and Kamara [16] and formulated for dynamic searchable symmetric encryption by Kamara et al. [33].

A variety of follow up work has been published in the area of searchable encryption, addressing further topics, e.g., performance improvements [15], searchable encryption in the public-key setting [10], additional functionality such as range queries [12], [19] and conjunctions [14], [31] or substring queries [20]. Note that many of these solutions are of great value for the pre-filtering process crucial for the performance of our approach. Fuller et al. [23] have published a comprehensive overview on cryptographically protected database operations recently. They identified join operation as important open-problem in order to translate all core database features to their encrypted counterparts.

b) Attribute Based Encryption: The property of our construction providing fine granular security for join operations is based on the additional blinding of join values implemented with attribute-based encryption (ABE) as introduced by Sahai et al. [50]. While the initial suggestions for ABE realized the idea of ciphertext-policy, Goyal et al. [24] published an orthogonal work for key-policy. Attrapadung et al. [6] published improvements decreasing the storage overhead induced by KP-ABE encrypted ciphertexts to be constant but with a fixed attribute domain size. Hohenberger and Waters [29] followed this approach and published a work with decreased decryption overhead requiring a constant number of bilinear pairings and gave a solution for an unlimited attribute universe. Due to big data applications we applied their scheme in order to minimize the amount of such computation expensive operations and to be flexible in the attribute domain. Attribute-hiding as property for ABE has been introduced by Katz et al., decreasing the information leakage by additional attribute privacy [34].

VI. Conclusion

We present a novel approach for cryptographically protected database joins with fine granular security. While schemes based on private set intersection require uniqueness of the join values per database table and alternative solutions without this constraint only provide all-or-nothing security, our scheme provides full flexibility and advanced protection. We leverage that most join queries on databases contain additional selection predicates. All previous schemes supporting secure joins unveil the complete inner-join result, leaking unnecessary information. Taking the additional selection process in consideration already during the encryption phase, our construction minimizes the information leakage of the join-operation by only unveiling the join values actually involved in the computation of the join's result set.

Our constructions is based on a combination of searchable symmetric encryption and key-policy attribute-based encryption, both applied as generic black boxes. Due to this black-box construction, our solution benefits from all further improvements in these active research areas. Further, our work comprises a formal security analysis and the practical feasibility is demonstrated by a prototypical implementation in a real system based on MySQL and Java.

References

[1] (2017) Ciphercloud. http://www.ciphercloud.com.

[2] (2017) Vaultive. https://vaultive.com.

[3] R. Agrawal, D. Asonov, M. Kantarcioglu, and Y. Li, "Sovereign joins," in *Proceedings of the 22nd International Conference on Data Engineering*, ser. ICDE, 2006.

[4] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," in *Proceedings of the ACM International Conference on Management of Data*, ser. SIGMOD, 2004.

[5] A. Arasu and R. Kaushik, "Oblivious query processing," *arXiv preprint arXiv:1312.4012*, 2013.

[6] N. Attrapadung, B. Libert, and E. De Panafieu, "Expressive key-policy attribute-based encryption with constant-size ciphertexts," in *International Workshop on Public Key Cryptography*, ser. WPKC, 2011.

[7] M. Bellare, A. Boldyreva, and A. O'Neill, "Deterministic and efficiently searchable encryption," in *Proceedings of the 27th International Conference on Advances in Cryptology*, ser. CRYPTO, 2007.

[8] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," *Advances in Cryptology*, 1998.

[9] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill, "Order-preserving symmetric encryption," in *Proceedings of the 28th International Conference on Advances in Cryptology*, ser. EUROCRYPT, 2009.

[10] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Advances in Cryptology*, ser. EUROCRYPT, 2004.

[11] D. Boneh, A. Sahai, and B. Waters, "Functional encryption: definitions and challenges," in *Proceedings of the 8th Conference on Theory of Cryptography*, ser. TCC, 2011.

[12] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proceedings of the 4th Theory of Cryptography Conference*, ser. TCC, 2007.

[13] B. Carbunar and R. Sion, "Toward private joins on outsourced data," *IEEE Transactions on Knowledge and Data Engineering*, 2012.

[14] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner, "Dynamic searchable encryption in very large databases: Data structures and implementation," in *Proc. of NDSS*, vol. 14, 2014.

[15] 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*, ser. CRYPTO, 2013.

[16] M. Chase and S. Kamara, "Structured encryption and controlled disclosure," in *Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security*, 2010.

[17] E. F. Codd, "A relational model of data for large shared data banks," *Communications of the ACM*, vol. 13, no. 6, 1970.

[18] 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*, ser. CCS, 2006.

[19] I. Demertzis, S. Papadopoulos, O. Papapetrou, A. Deligiannakis, and M. Garofalakis, "Practical private range search revisited," in *Proceedings of the ACM SIGMOD Conference on Management of Data*, ser. SIGMOD, 2016.

[20] 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*, ser. ESORICS, 2015.

[21] R. Fagin, M. Naor, and P. Winkler, "Comparing information without leaking it," *Communications of the ACM*, vol. 39, no. 5, 1996.

[22] M. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, ser. EUROCRYPT, 2004.

[23] 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 arXiv:1703.02014*, 2017.

[24] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM conference on Computer and communications security*, ser. CCS, 2006.

[25] P. Grubbs, K. Sekniqi, V. Bindschaedler, M. Naveed, and T. Ristenpart, "Leakage-abuse attacks against order-revealing encryption," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2017.

[26] 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*, ser. SIGMOD, 2002.

[27] H. Hacigumus, B. Iyer, and S. Mehrotra, "Providing database as a service," in *Proceedings of the 18th International Conference on Data Engineering*, ser. ICDE, 2002.

[28] F. Hahn and F. Kerschbaum, "Searchable encryption with secure and efficient updates," in *Proceedings of the 21st ACM Conference on Computer and Communications Security*, ser. CCS, 2014.

[29] S. Hohenberger and B. Waters, "Attribute-based encryption with fast decryption," in *Public-Key Cryptography*, ser. PKC, 2013.

[30] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation," in *Proceedings of the 19th Network and Distributed System Security Symposium*, ser. NDSS, 2012.

[32] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption," in *Proceedings of the 17th International Conference on Financial Cryptography and Data Security*, ser. FC, 2013.

[31] S. Kamara and T. Moataz, "Boolean searchable symmetric encryption with worst-case sub-linear complexity," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, ser. EUROCRYPT, 2017.

[33] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proceedings of the 19th ACM Conference on Computer and Communications Security*, ser. CCS, 2012.

[34] J. Katz, A. Sahai, and B. Waters, "Predicate encryption supporting disjunctions, polynomial equations, and inner products," in *Advances in Cryptology*, ser. EUROCRYPT, 2008.

[35] F. Kerschbaum, "Collusion-resistant outsourcing of private set intersection," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, ser. SAC, 2012.

[36] ——, "Outsourced private set intersection using homomorphic encryption," in *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS, 2012.

[37] F. Kerschbaum, M. Härterich, P. Grofig, M. Kohler, A. Schaad, A. Schröpfer, and W. Tighzert, "Optimal re-encryption strategy for joins in encrypted databases," in *IFIP Annual Conference on Data and Applications Security and Privacy*, ser. DBSEC, 2013.

[38] A. Kiayias, S. Papadopoulos, N. Triandopoulos, and T. Zacharias, "Delegatable pseudorandom functions and applications," in *Proceedings of the 20th ACM Conference on Computer and Communications Security*, ser. CCS, 2013.

[39] A. Lewko and B. Waters, "Decentralizing attribute-based encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2011, pp. 568–588.

[40] Y. Li and M. Chen, "Privacy preserving joins," in *IEEE 24th International Conference on Data Engineering*, ser. ICDE, 2008.

[41] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *Proceedings of the 27th USENIX Security Symposium*, ser. USENIX Security, 2018.

[42] Z. Liu, Z. Cao, and D. S. Wong, "Efficient generation of linear secret sharing scheme matrices from threshold access trees," IACR Cryptology ePrint Archive, Tech. Rep., 2010.

[43] 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*, ser. CCS, 2015.

[44] M. Naveed, M. Prabhakaran, and C. A. Gunter, "Dynamic searchable encryption via blind storage," in *Proceedings of the IEEE Symposium on Security and Privacy*, ser. S&P, 2014.

[45] O. Pandey and Y. Rouselakis, "Property preserving symmetric encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, ser. EUROCRYPT, 2012.

[46] H. Pang and X. Ding, "Privacy-preserving ad-hoc equi-join on outsourced data," *ACM Transactions on Database Systems*, 2014.

[47] B. Pinkas, T. Schneider, and M. Zohner, "Scalable private set intersection based on OT extension," Cryptology ePrint Archive, Report 2016/930, 2016, http://eprint.iacr.org/2016/895.

[48] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: protecting confidentiality with encrypted query processing," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, ser. SOSP, 2011.

[49] R. A. Popa and N. Zeldovich, "Cryptographic treatment of cryptdb's adjustable join," 2012. [Online]. Available: https://css.csail.mit.edu/cryptdb/

[50] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, ser. EUROCRYPT, 2005.

[51] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proceedings of the IEEE Symposium on Security and Privacy*, ser. S&P, 2000.

[52] E. Stefanov, C. Papamanthou, and E. Shi, "Practical dynamic searchable encryption with small leakage." vol. 2013, 2013.

[53] J. Van 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*, ser. USENIX Security, 2018.

[54] Y. Wang and H. Pang, "Probabilistic public key encryption for controlled equijoin in relational databases," *The Computer Journal*, 2016.

[55] M. Yung, "From mental poker to core business: why and how to deploy secure computation protocols?" in *Proceedings of the 22nd ACM Conference on Computer and Communications Security*, ser. CCS, 2015.