Towards More Secure Constructions of Adjustable Join Schemes

Shahram Khazaei, Mojtaba Rafiee

Abstract—An adjustable join (Adjoin) scheme [Popa-Zeldovich 2012] is a symmetric-key primitive that enables a user to securely outsource his database to a server, and later to issue join queries for a pair of columns. When queries are extended to a list of columns, `3Partition` security of Adjoin schemes [Mironov-Segev-Shahaf 2017] does not capture the expected security. To address this deficiency, we introduce the syntax and security notion of multi-adjustable join (M-Adjoin) schemes. We propose a new security notion for this purpose, which we refer to as `M3Partition`. The `3Partition` security of Adjoin extends to the `M3Partition` security of M-Adjoin in a straightforward way. The gap between `3Partition` and `M3Partition` is filled with a sequence $\{\texttt{M3P}_k\}_{k\in\mathbb{N}}$ of security definitions where $\texttt{M3P}_1$ and $\texttt{M3P}_\infty$, respectively, correspond to `3Partition` and `M3Partition`. We propose constructions for achieving both `M3Partition` and $\texttt{M3P}_k$ security levels. Our `M3Partition`-secure scheme joins m columns, each containing n elements, in time $\mathcal{O}(n^{m-1})$. Our $\texttt{M3P}_k$-secure scheme uses ideas from secret sharing in its construction and does the job in time $\mathcal{O}((m-1)n^k/k)$ with some leakage that we refer to as k-monotonous. It remains open if this barrier is inherent to the security definitions. Our schemes are substantially more efficient than previous ones.

Index Terms—Secure database outsourcing, Symmetric-key primitive, Join query, Monotonicity, Non-tranisitivity.

1 Introduction

There has been a surge in the usage of cloud services, especially storage and computing ones in recent years. In such settings, used by both enterprises and individuals, a user outsources his data to an external server. Over time, the user sends queries to the server and receives back the result of each one. The superiority of these services is that a user with limited computational and storage power can take advantage of the unlimited capabilities of the cloud server.

Database management systems (DBMS) are one of these services with great interests in industry and business. In such services, since there is no trust to the external servers, the databases are encrypted prior to outsourcing. CryptDB, designed by Popa et al. [1], [2], [3], [4], is one such notable system that supports a variety of SQL queries over encrypted databases. One of the most challenging issues in designing these services is supporting SQL queries, such as selections, projections, joins, aggregates, and orderings, on the encrypted database.

In this paper, we focus on the secure join queries on encrypted databases. Several research such as [4], [5], [6], [7], [8] have studied secure join queries and provided solutions with various trade-offs between security and efficiency. The scenario model for this functionality considers two main parties: a user and a server. The user outsources a database to the server, where a database contains a number of tables and each table includes several data records that are vertically partitioned into columns. When the user would like to issue a join query on his database, he generates a join token and sends it to the server. A join query is formulated as a pair of column labels. Finally, the server executes the requested join query on the encrypted database and returns the join result to the user.

The adjustable join scheme (Adjoin), first proposed by Popa and Zeldovich [4], is a symmetric-key primitive that supports the secure join queries for a pair of column labels on an encrypted database. The proposed security definition in [4] for this primitive does not capture transitivity leakage, and so it is far from the expected security. An adjustable join scheme has the transitive leakage, if for any three column labels l_i , l_j and l_k, join tokens for computing the joins between l_i and l_k and between l_k and l_j, allow to compute the join between l_i and l_j without asking his token.

Recently, Mironov et al. [8] proposed a strong and intuitive notion of security, called `3Partition`, for the adjustable join schemes, and argued that it indeed captures the security of such schemes (no transitive leakage). Also, they introduced natural simulation-based and indistinguishability-based notions that captured the minimal leakage of such schemes, and proved that the `3Partition` notion is positioned between their adaptive and non-adaptive variants with respect to some natural *minimal leakage*. The minimal leakage [9] reveals some accepted information such as the database dimensions (i.e total number of columns and the length of each column), the search pattern (i.e., the repetition of columns in different queries), the result pattern (i.e., the positions in which all columns of a join query contain identical elements) as well as the duplication pattern [8] (i.e., the positions in each column with identical contents for every column in the database).

In traditional applications of DBMS, the length of the join queries (m) are small, usually $m < 10$, although the databases allow for longer lengths (for example, the maximum length for the SQL Server is 256 [10]). Nevertheless, with the development of database applications such as: decision support system (DSS), online analytical processing (OLAP) and data mining (DM), join query lengths have also increased significantly [11]. On the other hand, the Adjoin schemes are designed for the join queries over a pair of columns, or multiple pairs of columns. However,

in the database outsourcing scenario model, since our goal is not to disclose any useful information about queries and database elements directly to the server, executing multiple pairs of columns instead of a list of columns reveals more information to the server. In other words, the previous security definitions for the Adjoin schemes have intermediate leakage and do not meet the expected security. We say an Adjoin scheme has the intermediate leakage, if the join token for a list of column labels allows to join a sub-list of column labels. Table 1, 2 and 3 show a simplified description of the transitive and intermediate leakages.

Let A and B be two tables. A $\bowtie$ B denotes join between tables A and B based on columns with the same labels. For simplicity, we have shown a selection of the join result as the output of the operator $\bowtie$ on the tables A and B.

TABLE 1: A simplified product database with tables on Orders, Customers and Shippers.

(a) Customers table.

CustID	CustName	PostalCode	PrefShipID
1	Maison Dewey	B-1180	100
2	Que Delícia	02389-673	200
3	Vaffeljernet	8200	300
4	Wilman Kala	21240	200

(b) Orders table.

OrdID	CustID	Date	ShipID
10	1	1996-07-04	100
20	2	1996-07-23	200
30	3	1996-08-14	200
40	4	1996-09-13	100

(c) Shippers table.

ShipID	ShipName	Phone
100	Speedy Express	(503) 555-9831
200	United Package	(503) 555-3199
300	Federal Shipping	(503) 555-9931

TABLE 2: Transitivity leakages for join queries (Customers $\bowtie$ Orders) and (Orders $\bowtie$ Shippers) using Adjoin scheme proposed by Popa and Zeldovich [4].

Expected result

(a) Customers $\bowtie$ Orders

CustID	CustName	OrdID	Date
1	Maison Dewey	10	1996-07-04
2	Que Delícia	20	1996-07-23

(b) Orders $\bowtie$ Shippers

OrdID	Date	ShipID	ShipName
10	1996-07-04	100	Speedy Express
20	1996-07-23	200	United Package
30	1996-08-14	200	United Package
40	1996-09-13	100	Speedy Express

Transitive leakage

(c) Customers $\bowtie$ Shippers

CustID	CustName	PrefShipID	ShipName
1	Maison Dewey	100	Speedy Express
2	Que Delícia	200	United Package
3	Vaffeljernet	300	Federal Shipping
4	Wilman Kala	200	United Package

TABLE 3: Intermediate leakages for join query (Customers $\bowtie$ Orders $\bowtie$ Shippers) using Adjoin scheme proposed by Mironov et al. [8].

Expected result

(a) Customers $\bowtie$ Orders $\bowtie$ Shippers

CustName	OrdID	Date	ShipName
Maison Dewey	10	1996-07-04	Speedy Express
Que Delícia	20	1996-07-23	United Package

Intermediate leakages

(b) Customers $\bowtie$ Orders

CustID	CustName	OrdID	Date
1	Maison Dewey	10	1996-07-04
2	Que Delícia	20	1996-07-23

(c) Orders $\bowtie$ Shippers

OrdID	CustID	ShipID	ShipName
10	1	100	Speedy Express
20	2	200	United Package
30	3	200	United Package
40	4	100	Speedy Express

In this paper, to capture the intermediate leakage, we propose the multi-adjustable join (M-Adjoin) schemes as an extension of the adjustable join schemes, and define a family $\{\texttt{M3P}_k\}_{k\in\mathbb{N}}$ of security notions, where an increase in parameter k reduces the leakage level.

We emphasize that an M-Adjoin scheme is a general and independent cryptographic primitive. Although our work is motivated by the secure join queries on the encrypted database, the multi-adjustable join schemes can be used by a variety of real world applications such as Boolean searchable symmetric encryption (BSSE) [12], private set intersection in the cloud scenarios (PSI) [13], privacy preserving data mining [14], and distributed storage systems [6].

1.1 Contributions

In this paper, we extend the notion of the adjustable join schemes to the *multi-adjustable join* (M-Adjoin) schemes, where the join queries are formulated as a list of column labels instead of a pair of column labels. We then show that unlike the Adjoin scheme, `3Partition` security is not sufficient for the M-Adjoin schemes. We conclude that an extension of `3Partition`, which we call `M3Partition`, is what we are looking for.

We define a family $\{\texttt{M3P}_k\}_{k\in\mathbb{N}}$ of security notions that fills the gap between `3Partition` and `M3Partition` security notions. More precisely, $\texttt{M3P}_1$ is exactly the `3Partition` security, $\texttt{M3P}_k$ positions between $\texttt{M3P}_{k-1}$ and $\texttt{M3P}_{k+1}$ but bellow `M3Partition`. We call a multi-adjustable join scheme *k-monotonous* if it allows an adversary to compute the join of an unqueried list of columns of size $k+1$ if it has already queried a superset of the list. A k-monotonous scheme makes it possible to compute the join of $m \geq k+1$ columns, each of length n, in time

$\mathcal{O}((m-1)n^k/k)$.

For every integer k, we propose an M-Adjoin construction with k-monotonous property. In particular, our construction with 1-monotonous property is more efficient than 1-monotonous constructions proposed in [8]. The size of adjustment token of our construction is m group elements and the previous ones are $4m$ and $2m$ group elements, where m is the number of columns in a join query. See Section 6 for detailed performance comparison and discussion.

Additionally, we propose another construction which is `M3Partition`-secure (and hence non-monotonous), but it requires $\mathcal{O}(n^{m-1})$ join time.

1.2 Paper organization

In Section 2, we provide notations and definitions that are required throughout this paper. Section 3 present the M-Adjoin syntax. In Section 4, the security definitions of M-Adjoin schemes are introduced. Our two proposed constructions for M-Adjoin, and their security proofs are presented in Section 5. The performance analysis for different M-Adjoin schemes is presented in Section 6. Finally, Section 7 concludes the paper and points out future directions.

2 PRELIMINARIES

2.1 Notation

Throughout the paper, we use $[m]$ to denote the set $\{1,\ldots,m\}$, where m is a positive integer. The security parameter is denoted by λ. Assuming that A is a (possibly) probabilistic algorithm, $y \leftarrow A(x)$ means that y is the output of A on input x. When A is a finite set, $x \leftarrow A$ stands for uniformly selecting an element x from A. We say that a function is negligible, if it is smaller than the inverse of any polynomial in λ for sufficiently large values of λ.

We let $\{0,1\}^\lambda$ denote the set of all strings of length λ, called words, and $\left(\{0,1\}^\lambda\right)^*$ denote the set of all finite lists of λ-bit long words. We use the notation w and l for denoting a word and a label, respectively, which for simplicity both[1] are considered to be λ-bit long (i.e., $w, l \in \{0,1\}^\lambda$). The labels are used to identify a column C in a database. Also, database columns are considered as a list of words (i.e., $C \in \left(\{0,1\}^\lambda\right)^*$). As a convention, we denote the output of a defined experiment by the experiment name itself.

2.2 Computational Indistinguishability

Let X_λ, Y_λ be distributions over $\{0,1\}^{l(\lambda)}$ for some polynomial $l(\lambda)$. We say that the families $\{X_\lambda\}$ and $\{Y_\lambda\}$ are computationally indistinguishable, and write $X_\lambda \approx Y_\lambda$, if for all probabilistic polynomial-time (PPT) distinguisher $\mathcal{D}$, there exists a negligible function ε such that

$$|\Pr[t \leftarrow X_\lambda : \mathcal{D}(t) = 1] - [t \leftarrow Y_\lambda : \mathcal{D}(t) = 1] \leq \varepsilon(\lambda).$$

For a pair of distributions X_λ and Y_λ, if $X_\lambda \approx Y_\lambda$ then for any PPT algorithms M, it holds that $M(X_\lambda) \approx M(Y_\lambda)$. This is known as the closure under efficient operations.

Let $X_1, X_2, \cdots, X_m$ be a sequence of probability distributions. Assume that the distinguisher $\mathcal{D}$ can distinguish between X_1 and X_m with advantage ε. Then, there exists some $i \in [1, \cdots, m-1]$ such that the distinguisher $\mathcal{D}$ can distinguish X_i and X_{i+1} with advantage $\frac{\varepsilon}{m}$. This is known as the hybrid lemma.

1. It is easy to remove this assumption and work with long messages and short labels as it is the case in practice. To keep our discussion simple, we stick to this conversion.

2.3 Basic primitives

2.3.1 Pseudorandom function

Let X, Y be two sets. A polynomial-time computable function $F : \{0,1\}^\lambda \times X \to Y$ is a pseudorandom function (PRF) if for every PPT adversary $\mathcal{A}$, the following quantity is negligible:

$$\mathsf{Adv}^{\mathsf{PRF}}_{F,\mathcal{A}}(\lambda) = |\Pr[k \leftarrow \{0,1\}^\lambda : \mathcal{A}^{F_k(\cdot)}(1^\lambda) = 1] - \Pr[f \leftarrow RF : \mathcal{A}^{f(\cdot)}(1^\lambda) = 1]|,$$

where RF is the set of all functions from X to Y.

2.3.2 Bilinear map:

Let $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ be cyclic groups of prime order q, and g_1, g_2 be generators for $\mathbb{G}_1$, $\mathbb{G}_2$, respectively. A bilinear map is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, which satisfies the following properties:

1) *Bilinearity:* $\forall x, y \in \mathbb{Z}_q :\ e(g_1^x, g_2^y) = e(g_1, g_2)^{xy}$,
2) *Non-degeneracy:* $e(g_1, g_2) \neq 1$,
3) *Computability:* e can be computed efficiently.

We assume that we have a PPT bilinear map generator $\mathcal{G}$ that on security parameter as input, outputs a tuple $Param = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, q, e)$.

2.4 Adjoin scheme

An adjustable join scheme (Adjoin), first introduced in [4], is a symmetric-key primitive that enables a client to generate an encoding of any word relative to any column label, and to generate a pair of tokens to compute the join of any two given columns.

Definition 1 (Adjoin syntax)**.** *An adjustable join scheme is a collection of four PPT algorithms Adjoin* $= (\mathsf{Gen}, \mathsf{Encod}, \mathsf{Token}, \mathsf{Adjust})$ *such that:*

- $(Param, K) \leftarrow \mathsf{Gen}(1^\lambda)$*: is a probabilistic key generation algorithm that takes as input a security parameter* λ*, and returns a secret key* K *and public parameters* $Param$*.*
- $\widetilde{w} \leftarrow \mathsf{Encod}_K(w, l)$*: is a deterministic encoding algorithm that takes as input a secret key* K*, a word* w *and a column label* l*, and outputs an encoded-word* $\widetilde{w}$*.*
- $(at_i, at_j) \leftarrow \mathsf{Token}_K(l_i, l_j)$*: is a probabilistic token generation algorithm that takes as input a secret key* K *and two column labels* (l_i, l_j)*, and returns a pair* (at_i, at_j) *of adjustment tokens.*
- $aew \leftarrow \mathsf{Adjust}_{Param}(\widetilde{w}, at)$*: is a deterministic algorithm that takes as input the public parameters* $Param$*, an encoded-word* $\widetilde{w}$ *and an adjustment token* at*, and outputs an adjusted encoded-word* aew*.*

Adjoin correctness and security. The correctness intuitively guarantees that no PPT adversary can find two column labels $l_i, l_j \in \{0,1\}^\lambda$ and two words $w \neq w' \in \{0,1\}^\lambda$ such that their adjusted encoded-words are the same, except with a negligible probability.

Adjoin security expresses that no useful information about the join queries and database elements is directly revealed. Mironov et al. [8] proposed a strong and intuitive notion of security, called `3Partition`, for the adjustable join schemes, and argued that it indeed captures the security of such schemes (no transitive leakage). To this end, they considered three disjoint groups of columns, and allowed join queries to be issued on these groups in a particular order. Since `3Partition` notion is also defined for the M-Adjoin schemes, to simplify and avoid duplication, we describe it in details in Section 4.

3 M-Adjoin syntax

A multi-adjustable join scheme (M-Adjoin) is a symmetric-key primitive that enables to generate an encoding of any word relative to any column label, and to generate a tuple of tokens enabling to compute the join of any given set of columns.

M-Adjoin schemes are used as follows. A user wishing to outsource his database to a server, first generates a secret key K and public parameters $Param$ using a key generation algorithm denoted by Gen. Then, the user computes an encoded-word $\widetilde{w}$ for every word w relative to any database column label l using an encoding algorithm denoted by Encod and sends them along with the public parameters $Param$ to the server. Later, when the user wants to send a join query $q = (l_1, \cdots, l_m)$ to the server, he computes a list of adjustment tokens $(at_1, \cdots, at_m)$ using a token generation algorithm denoted by Token. Upon receiving adjustment tokens $(at_1, \cdots, at_m)$, the server computes an adjusted word aw for every encoded-word relative to every column label in the join query using an adjustment algorithm denoted by Adjust. Finally, the server computes the result set from the adjusted words using an evaluation algorithm denoted by Eval, and sends them to the user. Below we formalize the primitive [2].

Definition 2 (M-Adjoin syntax)**.** *A multi-adjustable join scheme is a collection of five polynomial-time algorithms* $\Pi = (\mathsf{Gen}, \mathsf{Encod}, \mathsf{Token}, \mathsf{Adjust}, \mathsf{Eval})$ *such that:*

- $(Param, K) \leftarrow \mathsf{Gen}(1^\lambda)$*: is a probabilistic key generation algorithm that takes as input a security parameter* λ*, and returns a secret key* K *and public parameters* $Param$*.*
- $\widetilde{w} \leftarrow \mathsf{Encod}_K(w, l)$*: is a deterministic encoding algorithm that takes as input a secret key* K*, a word* w *and a column label* l*, and outputs an encoded-word* $\widetilde{w}$*.*
- $(at_1, \cdots, at_m) \leftarrow \mathsf{Token}_K(l_1, \cdots, l_m)$*: is a probabilistic token generation algorithm that takes as input a secret key* K *and a list of distinct column labels* $(l_1, \cdots, l_m)$*, and returns a tuple* $(at_1, \cdots, at_m)$ *of adjustment tokens.*
- $aw \leftarrow \mathsf{Adjust}_{Param}(\widetilde{w}, at)$*: is a deterministic algorithm that takes as input the public parameters* $Param$*, an encoded-word* $\widetilde{w}$ *and an adjustment token* at*, and outputs an adjusted word* aw*.*
- $b \leftarrow \mathsf{Eval}_{Param}(aw_1, \cdots, aw_m)$*: is a deterministic evaluation algorithm that takes as input the public parameters* $Param$ *and a list of adjusted words* $aw_1, \cdots, aw_m$*, and outputs a bit* b*.*

Correctness. *The scheme is said to be correct, if for any integer* $m \geq 2$*, any list of column labels* $(l_1, \cdots, l_m) \in (\{0,1\}^\lambda)^m$ *and any list of words* $(w_1, \cdots, w_m) \in (\{0,1\}^\lambda)^m$*, it holds that*

$$\mathsf{Adv}^{\mathsf{Cor}}_{\Pi}(\lambda) = \Pr\left[\begin{array}{l} (Param, K) \leftarrow \mathsf{Gen}(1^\lambda); \\ (at_1, \cdots, at_m) \leftarrow \mathsf{Token}_K(l_1, \cdots, l_m); \\ \forall i \in [m] \;\; \widetilde{w_i} \leftarrow \mathsf{Encod}_K(w_i, l_i); \\ \forall i \in [m] \;\; aw_i \leftarrow \mathsf{Adjust}_{Param}(\widetilde{w_i}, at_i) : \\ \quad \mathsf{Eval}(aw_1, \cdots, aw_m) = 1 \end{array}\right] \leq \varepsilon(\lambda),$$

if $w_i \neq w_j$ *for some distinct* $i, j \in [m]$*, and that the above probability is* 1 *if* $w_1 = \cdots = w_m$*.*

4 M-Adjoin security

In this section, we present the security definitions for the M-Adjoin schemes. These definitions can be classified in three categories: 1) the `3Partition` security notion, 2) the `M3Partition` security notion and 3) a family $\{\mathtt{M3P}_k\}_{k\in\mathbb{N}}$ of security notions. Each of these notions are first explained informally and then the formal definitions are provided.

4.1 The `3Partition` security notion

In this subsection, we adapt the `3Partition` security notion for the Adjoin [8] to M-Adjoin. Recall that the `3Partition` security has been proposed to capture the transitivity leakage (see Table 2).

The adversary of the `3Partition` security notion first defines three disjoint groups of columns, denoted by $\mathscr{L}$ (left), $\mathscr{M}$ (middle) and $\mathscr{R}$ (right). It can then adaptively receive encoded-word of every selected word relative to any chosen column label. The adversary can adaptively obtain the join tokens related to allowed queries. A query $q = (l_1, \cdots, l_m)$ is allowed if it is of one of the following two types:

T1) $(l_1, \cdots, l_m) \in \mathscr{L} \cup \mathscr{M}$ or,
T2) $(l_1, \cdots, l_m) \in \mathscr{M} \cup \mathscr{R}$.

The `3Partition` notion of security requires that such an adversary should not be able to compute the join of any list of column labels $(l_1, \cdots, l_m)$ such that $l_1, \cdots, l_m \in \mathscr{L} \cup \mathscr{R}$, $\{l_1, \cdots, l_m\} \cap \mathscr{L} \neq \emptyset$ and $\{l_1, \cdots, l_m\} \cap \mathscr{R} \neq \emptyset$. This is modeled by enabling the adversary to output a pair of challenge words w_0^*, w_1^*, and providing the adversary either with the encodings of w_0^* for all columns in $\mathscr{R}$ or with the encodings of w_1^* for all columns in $\mathscr{R}$. The adversary must be unable to distinguish these two cases with a non-negligible advantage, as long as the adversary did not explicitly ask for an encoding of w_0^* or w_1^* relative to some column label in $\mathscr{M} \cup \mathscr{R}$. Here is the formal definition.

Definition 3 (`3Partition` security)**.** *An M-Adjoin scheme such as* $\Pi = (\mathsf{Gen}, \mathsf{Encod}, \mathsf{Token}, \mathsf{Adjust}, \mathsf{Eval})$ *is* `3Partition`*-secure if for all PPT algorithms* $\mathcal{A}$*, there exists a negligible function* ε *such that*

$$|\Pr[\mathsf{Exp}^{\mathsf{3P}}_{\Pi,\mathcal{A}}(\lambda, 0) = 1] - \Pr[\mathsf{Exp}^{\mathsf{3P}}_{\Pi,\mathcal{A}}(\lambda, 1) = 1]| \leq \varepsilon(\lambda),$$

2. We remark that one can merge the Adjust and Eval algorithms into a single one since they are executed together by the server. However, for ease of notion, we stick to the convention in [3], [8] and consider two separate algorithms.

where for each $b \in \{0,1\}$, the experiment $\mathsf{Exp}^{\mathsf{3P}}_{\Pi,\mathcal{A}}(\lambda, b)$ is defined as follows:

Setup phase: *The challenger* Chal *samples $(Param, K) \leftarrow \mathsf{Gen}(1^\lambda)$, and initialize $\mathscr{L} = \mathscr{M} = \mathscr{R} = \emptyset$. The public parameters $Param$ are given as input to the adversary $\mathcal{A}$.*

Pre-challenge query phase: *The adversary $\mathcal{A}$ may adaptively issue* Addlbl, Encod *and* Token *queries, which are defined as follows:*

1) $\mathsf{Addlbl}(l, X)$*: adds the column label l to the group X, where $X \in \{\mathscr{L}, \mathscr{M}, \mathscr{R}\}$. The adversary $\mathcal{A}$ is not allowed to add a column label into more than one set (i.e., the groups $\mathscr{L}, \mathscr{M}$ and $\mathscr{R}$ must always be pairwise disjoint).*
2) $\mathsf{Encod}(w, l)$*: computes and returns an encoded-word $\widetilde{w} \leftarrow \mathsf{Encod}_K(w, l)$ to the adversary $\mathcal{A}$, where $l \in \mathscr{L} \cup \mathscr{M} \cup \mathscr{R}$.*
3) $\mathsf{Token}(l_1, \cdots, l_m)$*: computes and returns a list $(at_1, \cdots, at_m) \leftarrow \mathsf{Token}_K(l_1, \cdots, l_m)$ of adjustment tokens to the adversary $\mathcal{A}$, where $l_1, \cdots, l_m \in \mathscr{L} \cup \mathscr{M}$ or $l_1, \cdots, l_m \in \mathscr{M} \cup \mathscr{R}$.*

Challenge phase: *The adversary $\mathcal{A}$ chooses words w_0^* and w_1^* subject to the constraint that $\mathcal{A}$ did not previously issue a query of the form $\mathsf{Encod}_K(w, l)$ where $w \in \{w_0^*, w_1^*\}$ and $l \in \mathscr{M} \cup \mathscr{R}$. As a response, the adversary $\mathcal{A}$ obtains an encoded-word $\widetilde{w} \leftarrow \mathsf{Encod}_K(w_b^*, l)$ for every $l \in \mathscr{R}$.*

Post-challenge query phase: *As in the pre-challenge query phase, with the restriction that the adversary $\mathcal{A}$ is not allowed to issue a query of the form $\mathsf{Encod}(w, l)$, where $w \in \{w_0^*, w_1^*\}$ and $l \in \mathscr{M} \cup \mathscr{R}$. In addition, for each $\mathsf{Addlbl}(l, \mathscr{R})$ query, the adversary $\mathcal{A}$ is also provided with $\widetilde{w} \leftarrow \mathsf{Encod}_K(w_b^*, l)$.*

Output phase: *The adversary $\mathcal{A}$ outputs a value $\sigma \in \{0,1\}$ which is defined as the output of the experiment.*

4.2 The `M3Partition` and $\mathtt{M3P}_k$ security notions

Recall that the `3Partition`-security notion only requires that for any three disjoint sets $\mathscr{L}, \mathscr{M}, \mathscr{R}$, the ability to compute the joins in $\mathscr{L} \cup \mathscr{M}$ and $\mathscr{M} \cup \mathscr{R}$ does not allow to compute the join between any column in $\mathscr{L}$ and any column in $\mathscr{R}$. However, this notion has intermediate leakage and does not meet the sufficient and expected security (see Table 3). The reason is that the token generation algorithm may leak some undesirable information to the adversary without violating the `3Partition` (and more generally $\mathtt{M3P}_k$) security. To see how this could happen, consider an M-Adjoin scheme that has the following property.

Definition 4 (k-monotonicity property)**.** *Let k be an integer. We say that an M-Adjoin scheme has the k-monotonicity property, or it is k-monotonous, if the following holds: for every $m \geq k+1$, for every query $q = (l_1, \cdots, l_m)$, for every valid token $(at_1, \cdots, at_m)$ for q, and for every subset $A \subseteq [m]$ of size at least $k+1$, it holds that $(at_i)_{i \in A}$ is a valid token for the query $(l_i)_{i \in A}$.*

The k-monotonicity property allows an adversary to compute the join of an unqueried list of size $k+1$ if it has already queried a superset of the list of size at least $k+2$. For our convenience, we define a weaker version of the monotonicity property.

Definition 5 (Weak k-monotonicity property)**.** *It is the same as the k-monotonicity property except that the condition "for every subset $A \subseteq [m]$" is replaced with "for some subset $A \subseteq [m]$".*

An illustrative example. Consider an M-Adjoin scheme with the 2-monotonicity property. Suppose an adversary obtains a tuple $(at_1, \cdots, at_5)$ of adjustment tokens generated by the Token algorithm for join query $(l_1, \cdots, l_5)$. In this case, the adversary can extract the valid tokens listed in Table 4 from the tuple $(at_1, \cdots, at_5)$ without requesting them.

TABLE 4: Valid extractable tokens.

(at_1, at_2, at_3)	(at_1, at_2, at_4)	(at_1, at_2, at_5)	(at_1, at_3, at_4)
(at_1, at_3, at_5)	(at_1, at_4, at_5)	(at_2, at_3, at_4)	(at_2, at_3, at_5)
(at_2, at_4, at_5)	(at_3, at_4, at_5)		

The M-Adjoin schemes of [8] have the 1-monotonicity property with the adjustment token size $\mathcal{O}(m)$. For every integer k, we propose a k-monotonous scheme with adjustment token size $O(m)$ (the difference in the hidden constant factor is huge). It is easily seen that a k-monotonous scheme makes it possible to compute the join of $m > k+1$ (resp. $2 \leq m \leq k+1$) columns, each of length n, in time $\mathcal{O}(\lceil \frac{m-1}{k} \rceil n^k)$ (resp. $\mathcal{O}(n^{m-1})$). The security of such schemes is captured by the $\mathtt{M3P}_k$-security. Below, we provide a detailed description of this time complexity.

For a k-monotonous scheme with $\mathtt{M3P}_k$ security level, since we can get the join results for each $k+1$ column, we first make $\left\lceil \frac{(m-1)}{k} \right\rceil$ sub-lists, each of length $k+1$. Then for each sub-list, we first invoke Algorithm Adjust for all elements of all the columns included in the sub-list, and then invoke Algorithm Eval to check the equality of elements. Therefore, to compute the join of m columns, each of length n, Algorithm Adjust is invoked $\left(\left\lceil \frac{(m-1)}{k} \right\rceil (k+1)n\right)$ times and Algorithm Eval is invoked $\left(\left\lceil \frac{(m-1)}{k} \right\rceil n^k\right)$ times. Hence,the time complexity of the join queries is $\mathcal{O}(\lceil \frac{m-1}{k} \rceil n^k)$.

4.2.1 Informal definition

Our previous discussions leads us towards a new security definition which we refer to as the `M3Partition`-security. Its experiment considers an adversary similar to the `3Partition` experiment but with less constraints on the join tokens. Again, the adversary adaptively defines three disjoint groups of columns, denoted by $\mathscr{L}$ (left), $\mathscr{M}$ (middle) and $\mathscr{R}$ (right). It adaptively requests an encoded-word of his selected word relative to any column and join tokens related to allowed queries. Recall that in the `3Partition` experiment a join query $q = (l_1, \cdots, l_m)$ was allowed to be of any of the following two types:

T1) $l_1, \cdots, l_m \in \mathscr{L} \cup \mathscr{M}$ or,
T2) $l_1, \cdots, l_m \in \mathscr{M} \cup \mathscr{R}$.

Here we further allow the adversary to issue the following third type:

T3) $l_1, \cdots, l_m \in \mathscr{L} \cup \mathscr{M} \cup \mathscr{R}$ and $\{l_1, \cdots, l_m\} \cap \mathscr{M} \neq \emptyset$.

The game then continues as in the `3Partition` experiment.

For every integer k, we define the $\mathtt{M3P}_k$ security by modifying the third type of allowed queries as follows:

T3′) $l_1, \cdots, l_m \in \mathscr{L} \cup \mathscr{M} \cup \mathscr{R}, \{l_1, \cdots, l_m\} \cap \mathscr{M} \neq \emptyset$ and $m \leq k+1$.

That is, the query length must be at most $k+1$. Notice that when $k = 1$, the allowed queries of third type are essentially those of the first and second types; i.e., the experiment is exactly the `3Partition` experiment. The `M3Partition` experiment can be viewed as the limit of the $\mathtt{M3P}_k$ experiment when k goes to infinity. Therefore, we use `M3Partition` and $\mathtt{M3P}_\infty$ interchangeably.

4.2.2 Formal definition

Below, we provide a formal definition of the `M3Partition` and $\mathtt{M3P}_k$ security notions.

Definition 6 ($\mathtt{M3P}_k$ security, $k \in \mathbb{N} \cup \{\infty\}$)**.** *Let $k \in \mathbb{N} \cup \{\infty\}$. An M-Adjoin scheme such as $\Pi = (\mathsf{Gen}, \mathsf{Encod}, \mathsf{Token}, \mathsf{Adjust}, \mathsf{Eval})$ is $\mathtt{M3P}_k$-secure if for all PPT algorithms $\mathcal{A}$, there exists a negligible function ε such that*

$$|\Pr[\mathsf{Exp}^{\mathtt{M3P}_k}_{\Pi,\mathcal{A}}(\lambda,0)] - \Pr[\mathsf{Exp}^{\mathtt{M3P}_k}_{\Pi,\mathcal{A}}(\lambda,1)]| \leq \varepsilon(\lambda),$$

where for each $b \in \{0,1\}$, the experiment $\mathsf{Exp}^{\mathtt{M3P}_k}_{\Pi,\mathcal{A}}(\lambda, b)$ is defined as follows:

Setup phase: *The challenger* Chal *samples $(Param, K) \leftarrow \mathsf{Gen}(1^\lambda)$, and initialize $\mathscr{L} = \mathscr{M} = \mathscr{R} = \emptyset$. The public parameters $Param$ are given as input to the adversary $\mathcal{A}$.*

Pre-challenge query phase: *The adversary $\mathcal{A}$ may adaptively issue* Addlbl, Encod *and* Token *queries, which are defined as follows:*

1) $\mathsf{Addlbl}(l, X)$*: adds the column label l to the group X, where $X \in \{\mathscr{L}, \mathscr{M}, \mathscr{R}\}$. The adversary $\mathcal{A}$ is not allowed to add a column label into more than one set (i.e., the groups $\mathscr{L}, \mathscr{M}$ and $\mathscr{R}$ must always be pairwise disjoint).*
2) $\mathsf{Encod}(w, l)$*: computes and returns an encoded-word $\widetilde{w} \leftarrow \mathsf{Encod}_K(w, l)$ to the adversary $\mathcal{A}$, where $l \in \mathscr{L} \cup \mathscr{M} \cup \mathscr{R}$.*
3) $\mathsf{Token}(l_1, \cdots, l_m)$*: computes and returns a list $(at_1, \cdots, at_m) \leftarrow \mathsf{Token}_K(l_1, \cdots, l_m)$ of adjustment tokens to the adversary $\mathcal{A}$, where*
   - *$l_1, \cdots, l_m \in \mathscr{L} \cup \mathscr{M}$,*
   - *or $l_1, \cdots, l_m \in \mathscr{M} \cup \mathscr{R}$,*
   - *or $l_1, \cdots, l_m \in \mathscr{L} \cup \mathscr{M} \cup \mathscr{R}$, $\{l_1, \cdots, l_m\} \cap \mathscr{M} \neq \emptyset$ and $m \leq k+1$.*

Challenge phase: *The adversary $\mathcal{A}$ chooses a pair of challenge words w_0^* and w_1^* subject to the constraint that $\mathcal{A}$ did not previously issue a query of the form $\mathsf{Encod}(w, l)$ where $w \in \{w_0^*, w_1^*\}$ and $l \in \mathscr{M} \cup \mathscr{R}$. As a response, the adversary $\mathcal{A}$ obtains an encoded-word $\widetilde{w}^* \leftarrow \mathsf{Encod}_K(w_b^*, l)$ for every $l \in \mathscr{R}$.*

Post-challenge query phase: *As in the pre-challenge query phase, with the restriction that the adversary $\mathcal{A}$ is not allowed to issue a query of the form $\mathsf{Encod}_K(w, l)$, where $w \in \{w_0^*, w_1^*\}$ and $l \in \mathscr{M} \cup \mathscr{R}$. In addition, for each $\mathsf{Addlbl}(l, \mathscr{R})$ query, the adversary $\mathcal{A}$ is also provided with $\widetilde{w} \leftarrow \mathsf{Encod}_K(w_b^*, l)$.*

Output phase: *The adversary $\mathcal{A}$ outputs a value $\sigma \in \{0,1\}$ which is defined as the output of the experiment.*

Definition 7 (`M3Partition` security)**.** *An $\mathtt{M3P}_\infty$-secure M-Adjoin scheme is simply called `M3Partition`-secure.*

4.3 Security relations

The following corollary trivially follows by security definitions.

Corollary 8 (Trivial implications)**.** *Let k be an integer. Then,*

(a) **($\mathtt{M3Partition} \implies \mathtt{M3P}_k$)** *Any `M3Partition`-secure M-Adjoin scheme is $\mathtt{M3P}_k$-secure, too.*
(b) **($\mathtt{M3P}_{k+1} \implies \mathtt{M3P}_k$)** *Any $\mathtt{M3P}_{k+1}$-secure M-Adjoin scheme is $\mathtt{M3P}_k$-secure, too.*
(c) **(Limit cases)** *$\mathtt{M3P}_1 \equiv$ `3Partition` and $\mathtt{M3P}_\infty \equiv$ `M3Partition`.*

The following lemma is useful for a separation between `M3Partition` and $\mathtt{M3P}_k$ and between $\mathtt{M3P}_k$ and $\mathtt{M3P}_{k+1}$.

Lemma 9 (Weak k-monotonicity $\implies \sim \mathtt{M3P}_{k+1}$)**.** *An M-Adjoin scheme with the weak k-monotonicity property is not $\mathtt{M3P}_{k+1}$-secure.*

Proof. Without loss of generality assume that the weak k-monotonicity property holds for the set $A = \{1, \cdots, k+1\}$. In the pre-challenge phase, the adversary chooses $k+2$ distinct labels $\ell_1, \ldots, \ell_{k+2}$ at random and issues the following queries: $\mathsf{Addlbl}(\ell_i, \mathscr{L})$, for every $i \in [k]$, $\mathsf{Addlbl}(\ell_{k+1}, \mathscr{R})$ and $\mathsf{Addlbl}(\ell_{k+2}, \mathscr{M})$. In the challenge phase, he chooses a pair of distinct challenge word (w_0^*, w_1^*) at random and receives an encoded-word $\widetilde{w}_{k+1} = \mathsf{Encod}_K(w_b^*, \ell_{k+1})$, for some randomly chosen $b \in \{0,1\}$, which is unknown to him. In the post-challenge phase, he issues the queries $\widetilde{w_i} = \mathsf{Encod}_K(w_0^*, \ell_i)$ for every $i \in [k]$. He also requests the adjustment token $(at_1, \ldots, at_{k+2})$ for the query $(\ell_1, \ldots, \ell_{k+2})$. By the weak k-monotonicity property, $(at_1, \ldots, at_{k+1})$ is a valid adjustment token for the query $(\ell_1, \ldots, \ell_{k+1})$. The adversary can then determine if $w_0^* = \ldots = w_0^* = w_b^*$ by executing the adjustment algorithm and then the evaluation algorithm; that is, he learns b and outputs it. Therefore, his advantage in the $\mathtt{M3P}_{k+1}$ experiment is 1. □

Proposition 10 (Separations)**.** *Let k be an integer. Then,*

(a) **($\mathtt{M3P}_k \not\implies \mathtt{M3P}_{k+1}$)** *An $\mathtt{M3P}_k$-secure M-Adjoin scheme is not necessarily $\mathtt{M3P}_{k+1}$-secure.*
(b) **($\mathtt{M3P}_k \not\implies$ `M3Partition`)** *An $\mathtt{M3P}_k$-secure M-Adjoin scheme is not necessarily `M3Partition`-secure.*

Proof. Let Π be an $\mathtt{M3P}_k$-secure M-Adjoin scheme. We modify Π to get a scheme $\widetilde{\Pi}$ which is weakly k-monotonous but retains its $\mathtt{M3P}_k$-security. By Lemma 9, it is not $\mathtt{M3P}_{k+1}$-secure, proving Part (a). Part (b) follows by Lemma 9.

The key generation algorithms of Π and $\widetilde{\Pi}$ are the same. Let $q = (l_1, \ldots, l_m)$ be a query that is given to the token generation algorithm of $\widetilde{\Pi}$. A token $(at_1, \ldots, at_m)$ for q is first computed using the token generation algorithm of Π which will be the output of if $m \leq k+1$. Otherwise, a token $(at'_1, \ldots, at'_{k+1})$ is also generated for $q' = (l_1, \ldots, l_{k+1})$ using Π and the adjustment token of q in $\widetilde{\Pi}$ will be

$$\big((at_1, at'_1), \ldots, (at_{k+1}, at'_{k+1}), at_{k+2}, \cdots, at_m\big)\ .$$

The other algorithms are modified accordingly. The weak k-monotonicity property of $\widetilde{\Pi}$ is clear.

It remains to prove that the modified scheme remains $\mathtt{M3P}_k$-secure. To see this, recall the constraints in the $\mathtt{M3P}_k$

experiment. The adversary is allowed to issue a join query $q = (\ell_1, \cdots, \ell_m)$ of one of the following types:

T1) $l_1, \cdots, l_m \in \mathscr{L} \cup \mathscr{M}$ or,
T2) $l_1, \cdots, l_m \in \mathscr{M} \cup \mathscr{R}$.
T3) $l_1, \cdots, l_m \in \mathscr{L} \cup \mathscr{M} \cup \mathscr{R}$, $\{l_1, \cdots, l_m\} \cap \mathscr{M} \neq \emptyset$ and $m \leq k+1$.

Notice that our modification (i.e., including an adjustment token for the sub-query $q' = (\ell_1, \ldots, \ell_{k+1})$ when $m \geq k+2$) does not provide anything new to adversary since he was already allowed to issue such a query. □

5 Our M-Adjoin constructions

In this section, we present two M-Adjoin schemes. The first construction is M3Partition secure (and hence, non-monotonous) but the second one is k-monotonous where $k \geq 1$ is an arbitrary parameter but it is $\mathtt{M3P}_{k+1}$-secure

We use a bilinear group generator $\mathcal{G}$ that takes as input the security parameter λ, and outputs a tuple $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, q, e)$, where $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ are cyclic groups of prime order q, and g_1, g_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. The mapping $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerate efficiently computable bilinear map. We also use a pseudo-random function $F : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \mathbb{Z}_q^*$.

5.1 A non-monotonous scheme

Our main scheme is non-monotonous and indeed it is M3Partition-secure assuming the truth of a new computational hardness assumption, called MXDHV (Assumption 11), which is a variant of the XDH assumption [15], [16], [17], [18]. The algorithms of our main scheme $\Pi = (\mathsf{Gen}, \mathsf{Encod}, \mathsf{Token}, \mathsf{Adjust})$ are defined as follows:

- $(\mathbf{Param}, \mathbf{K} = (\mathbf{lk}, \mathbf{wk})) \leftarrow \mathsf{Gen}(\mathbf{1}^\lambda)$: It runs $Param \leftarrow \mathcal{G}(1^\lambda)$, and chooses a label-key $lk \in \{0,1\}^\lambda$ and a word-key $wk \in \{0,1\}^\lambda$ uniformly at random.
- $\widetilde{\mathbf{w}} \leftarrow \mathsf{Encod}_{\mathbf{K}}(\mathbf{w}, \mathbf{l})$: It computes encoded-word as $\widetilde{w} := g_1^{F_{lk}(l) \cdot F_{wk}(w)}$.
- $(\mathbf{at_1}, \cdots, \mathbf{at_m}) \leftarrow \mathsf{Token}_{\mathbf{K}}(\mathbf{l_1}, \cdots, \mathbf{l_m})$: It chooses random values $r_1, \cdots, r_m \in \mathbb{Z}_q$ subject to $r_1 + \cdots + r_m = 0$ and computes a list of adjustment tokens $(at_1, \cdots, at_m)$ as follows $at_i = g_2^{\frac{P}{F_{lk}(l_i)} \cdot r_i}$, where $P = \prod_{j=1}^{m} F_{lk}(l_j)$.
- $\mathbf{aw} \leftarrow \mathsf{Adjust}_{\mathbf{Param}}(\widetilde{\mathbf{w}}, \mathbf{at})$: It computes the adjusted word as $aw = e(\widetilde{w}, at) \in \mathbb{G}_T$.
- $\mathbf{b} \leftarrow \mathsf{Eval}_{\mathbf{Param}}(\mathbf{aw_1}, \cdots, \mathbf{aw_m})$: It outputs 1 if and only if $\prod_{i=1}^{m} aw_i = 1$.

Correctness. For any integer $m \geq 2$, any list of column labels $(l_1, \cdots, l_m) \in (\{0,1\}^\lambda)^m$ and any list of words $(w_1, \cdots, w_m) \in (\{0,1\}^\lambda)^m$, it holds that

$$aw_j = \mathsf{Adjust}_{Param}(\mathsf{Encod}_K(w_j, l_j), at_j)$$
$$= e(g_1^{F_{lk}(l_j) \cdot F_{wk}(w_j)}, g_2^{\frac{P}{F_{lk}(l_j)} \cdot r_j}) = e(g_1, g_2)^{F_{wk}(w_j) \cdot P \cdot r_j}, \quad (1)$$

for every $j \in [m]$, where $P = \prod_{j=1}^{m} F_{lk}(l_j)$, $(Param, K)$ is the output of $\mathsf{Gen}(1^\lambda)$, $(at_1, \cdots, at_m)$ is the output of $\mathsf{Token}_K(l_1, \cdots, l_m)$, and $r_1, \cdots, r_m$ are random values from $\mathbb{Z}_q$ subject to $r_1 + \cdots + r_m = 0$. Therefore, if $w_1 = \cdots = w_m$ then the following equality always holds $\prod_{i=1}^{m} aw_i = 1$.

Moreover, if $w_i \neq w_j$ for some distinct $i, j \in [m]$, then with an overwhelming probability $F_{wk}(w_i) \neq F_{wk}(w_j)$, since F is a pseudo-random function. It is easy to show that $\mathsf{Adv}^{\mathsf{Cor}}_{\text{M-Adjoin}}(\lambda)$, that is the probability that $\prod_{i=1}^{m} aw_i = 1$, is at most $\frac{2}{q} + \varepsilon(\lambda)$, where $\varepsilon(\lambda)$ is some negligible function.

5.2 A k-monotonous scheme

For every integer k, we present a modified version of our non-monotonous scheme which is $\mathtt{M3P}_k$-secure assuming the truth of MXDHV assumption (Assumption 11). All the algorithms are exactly the same as that of the non-monotonous scheme, except the token generation and evaluation algorithms. Let us give an intuition of the required modifications.

The token generation algorithm of the main scheme chooses m random values $r_1, \cdots, r_m \in \mathbb{Z}_q$ subject to $r_1 + \cdots + r_m = 0$. One can view it as a simple (m, m)-threshold secret sharing of the value $0 \in \mathbb{Z}_q$. By using an $(m, k+1)$-threshold scheme we will get what we want. Here are the modified algorithms:

- $(\mathbf{at_1}, \cdots, \mathbf{at_m}) \leftarrow \mathsf{Token}_{\mathbf{K}}(\mathbf{l_1}, \cdots, \mathbf{l_m})$: If $m \geq k+1$, then share the value $0 \in \mathbb{Z}_q$ using a linear $(m, k+1)$-threshold scheme (such as Shamir's) to get the shares $r_1, \cdots, r_m \in \mathbb{Z}_q$. If $2 \leq m \leq k$, then share the value $0 \in \mathbb{Z}_q$ using a linear (m, m)-threshold scheme to get the shares. Then, output a list of adjustment tokens $(at_1, \cdots, at_m)$ as before.
- $\mathbf{b} \leftarrow \mathsf{Eval}_{\mathbf{Param}}(\mathbf{aw_1}, \cdots, \mathbf{aw_m})$: If $m \geq k+1$, then output 1 if and only if $\prod_{i=1}^{k+1} aw_{i+j}^{\alpha_{i+j}} = 1$, for every $j = 0, k, 2k, 3k, \ldots, \lceil \frac{m-1}{k} \rceil k$, where $(\alpha_{i_1}, \ldots, \alpha_{i_{k+1}}) \in \mathbb{Z}_q^{k+1}$ are the (fixed) coefficients that makes it possible to compute the secret (i.e., 0) from the shares $(r_{i_1}, \ldots, r_{i_{k+1}})$. If $2 \leq m \leq k$, then output 1 if and only if $\prod_{i=1}^{m} aw_i^{\alpha_i} = 1$.

Correctness. The proof is similar to the correctness of the non-monotonous scheme. As we saw above in Equation (1), for any integer $m \geq 2$, any list of column labels $(l_1, \cdots, l_m) \in (\{0,1\}^\lambda)^m$ and any list of words $(w_1, \cdots, w_m) \in (\{0,1\}^\lambda)^m$, it holds that $aw_j = e(g_1, g_2)^{F_{wk}(w_j) \cdot P \cdot r_j}$, where here $r_1, \cdots, r_m \in \mathbb{Z}_q$ are the shares generated by the threshold secret sharing scheme that correspond to the secret value 0.

When $m \geq k+1$ and a $(m, k+1)$-threshold scheme is used, for every subset $A = \{i_1, \ldots, i_{k+1}\} \subset \{1 \ldots, m\}$, we have $\alpha_{i_1} r_{i_1} + \ldots + \alpha_{i_{k+1}} r_{i_{k+1}} = 0$. Therefore, if $w_1 = \cdots = w_m$ then we have:

$$\prod_{i=1}^{k+1} aw_{i+j}^{\alpha_{i+j}} = 1, \ j = 0, k, 2k, 3k, \ldots, \lceil \frac{m-1}{k} \rceil k \, .$$

Moreover, if $w_1, \ldots, w_m$ are not all the same, then there exists some $j \in \{0, k, 2k, 3k, \ldots, \lceil \frac{m-1}{k} \rceil k\}$ and $i, i' \in \{1, \ldots, k+1\}$ such that $w_{j+i} \neq w_{j+i'}$. Consequently, with an

overwhelming probability $F_{wk}(w_{j+i}) \neq F_{wk}(w_{j+i'})$, since F is a pseudo-random function. It is easy to show that the probability that $\prod_{i=1}^{k+1} aw_{i+j}^{\alpha_{i+j}} = 1$, is at most $\frac{2}{q} + \varepsilon(\lambda)$, where $\varepsilon(\lambda)$ is some negligible function. Therefore, $\mathsf{Adv}^{\mathsf{Cor}}_{\text{M-Adjoin}}(\lambda) \leq \frac{2}{q} + \varepsilon(\lambda)$ is negligible.

The case where $2 \leq m \leq k$ and a (m, m)-threshold scheme is used is similar.

5.3 Security analysis

The `M3Partition`-security of our non-monotonous scheme and the $\mathtt{M3P}_k$-security of our k-monotonous scheme both rely on a new computational hardness assumption, that we call the mixed external Diffie-Hellman variant (MXDHV) assumption, which is a variant of the XDH assumption, formalized in [15], [16], [17], [18].

Assumption 11. *MXDHV assumption for the bilinear map generator $\mathcal{G}$ states that it is hard to distinguish $g_1^{cm_0}$ from a random group element g_1^r given random group elements $g_1^a, g_1^c, g_1^{am_0}, g_2^{r'}, g_2^{r''}, g_2^{ar'}, g_2^{cr''}, g_2^{acr'}$ and $g_2^{acr''}$.*

We have examined the validity of our new assumption in the *group generic model*. The generic group model, introduced by Shoup [19], provides some confidence on the hardness of group-based hardness assumptions in cryptography. In this model, group elements are encoded by random bit-strings and the adversary has oracle access to some basic operations (e.g., multiplications and inversions). Very recently, Barthe et al. [20] have provided an automated tool which takes as input an assumption and outputs either a proof or an algebraic attack against the assumption. We have tested validity of our hardness assumption using an implementation of this method [3].

Theorem 12. *Suppose that F is a pseudo-random function and the MXDHV assumption holds relative to $\mathcal{G}$. Then,*

- *The proposed non-monotonous construction of Section 5.1 is `M3Partition`-secure.*
- *The proposed k-monotonous construction of Section 5.2 is $\mathtt{M3P}_k$-secure.*

Proof. The proof of both claims are quite similar and the differences will be made clear in the course of our argument.

For proving the first claim, let Π denote the non-monotonous M-Adjoin construction of Section 5.1 and let $\mathsf{Exp}_{\Pi,\mathcal{A}}(\lambda, b)$ denote the experiment $\mathsf{Exp}^{\mathtt{M3P}^\infty}_{\Pi,\mathcal{A}}(\lambda, b)$ where $\mathcal{A}$ is an adversary, λ is the security parameter and $b \in \{0, 1\}$.

For the second claim, Π denotes the k-monotonous scheme of Section 5.2 and $\mathsf{Exp}_{\Pi,\mathcal{A}}(\lambda, b)$ stands for the experiment $\mathsf{Exp}^{\mathtt{M3P}_k}_{\Pi,\mathcal{A}}(\lambda, b)$.

We need to show that $\mathsf{Exp}_{\Pi,\mathcal{A}}(\lambda, 0) \approx_c \mathsf{Exp}_{\Pi,\mathcal{A}}(\lambda, 1)$, for every PPT adversary $\mathcal{A}$. Let $\mathsf{Exp}_{\$\mathsf{F},\mathcal{A}}(\lambda, b)$ denote the experiment obtained from $\mathsf{Exp}_{\Pi,\mathcal{A}}(\lambda, b)$ by replacing the pseudo-random functions F_{lk} and F_{wk} with truly random functions f and h, respectively. By the pseudo-randomness property of F, it holds that the advantage of the adversary in distinguishing between the experiments $\mathsf{Exp}_{\Pi,\mathcal{A}}(\lambda, b)$ and $\mathsf{Exp}_{\$\mathsf{F},\mathcal{A}}(\lambda, b)$ is negligible, for $b = 0, 1$. Therefore, to prove the `M3Partition` security of the Π scheme, it is sufficient to show that $\mathsf{Exp}_{\$\mathsf{F},\mathcal{A}}(\lambda, 0) \approx_c \mathsf{Exp}_{\$\mathsf{F},\mathcal{A}}(\lambda, 1)$.

By hybrid lemma and under the MXDHV assumption, it holds that

$$X_\lambda \triangleq (g_1^a, g_1^c, g_1^{am_0}, g_1^{am_1}, g_1^{cm_0}, g_2^{r'}, g_2^{r''}, g_2^{ar'}, g_2^{cr''}, g_2^{acr'}, g_2^{acr''}) \approx_c (g_1^a, g_1^c, g_1^{am_0}, g_1^{am_1}, g_1^{cm_1}, g_2^{r'}, g_2^{r''}, g_2^{ar'}, g_2^{cr''}, g_2^{acr'}, g_2^{acr''}) \triangleq Y_\lambda,$$

where $Param = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, q, e)$ is the output of $\mathcal{G}(1^\lambda)$ and $a, c, m_0, m_1, r, r', r''$ are independently and uniformly chosen from $\mathbb{Z}_q^*$. For simplicity, we suppose that during the pre-challenge query phase, the adversary $\mathcal{A}$ does not issue a query of the form $\mathsf{Encod}_K(w_0^*, l)$ or $\mathsf{Encod}_K(w_1^*, l)$, from any column label l, where w_0^* and w_1^* are the challenge words. Similar to [8], we handle this exception at the end of the proof. We claim that there exists a polynomial-time challenger (or distinguisher) Chal, with oracle access to $\mathcal{A}$, such that it holds that $\mathsf{Chal}^{\mathcal{A}}(X_\lambda) \equiv \mathsf{Exp}_{\$\mathsf{F},\mathcal{A}}(\lambda, 0)$ and $\mathsf{Chal}^{\mathcal{A}}(Y_\lambda) \equiv \mathsf{Exp}_{\$\mathsf{F},\mathcal{A}}(\lambda, 1)$. Given a sample X_λ or Y_λ as an input, and $\mathcal{A}$ as an oracle, the challenger Chal manages to simulate the random functions f and h as follows (without knowing a, c, m_0 and m_1 explicitly):

$$f(l) = \begin{cases} a \cdot f_l & l \in \mathscr{L} \\ f_l & l \in \mathscr{M} \\ c \cdot f_l & l \in \mathscr{R} \end{cases}, \quad h(w) = \begin{cases} m_0 & w = w_0^* \\ m_1 & w = w_1^* \\ h_w & w \neq w_0^*, w_1^* \end{cases}, \quad (2)$$

where f_l and h_w are randomly chosen elements of $\mathbb{Z}_q^*$, and w_0^* and w_1^* are the challenge words. In Appendix A, we describe the challenger in details.

Since we assumed that the adversary $\mathcal{A}$ does not query w_0^* or w_1^* in the pre-challenge query phase, we suppose that $h_{w_0} = m_0$ and $h_{w_1} = m_1$. Therefore, in the case that the challenger is given a sample of X_λ as an input, the challenger responds the challenge with encoded-word of w_0^*, so we get the experiment $\mathsf{Exp}_{\$\mathsf{F},\mathcal{A}}(\lambda, 0)$. Similarly, in the case the challenger is given as input a sample of Y_λ, we get the experiment $\mathsf{Exp}_{\$\mathsf{F},\mathcal{A}}(\lambda, 1)$.

We now consider the general case where the adversary $\mathcal{A}$ may query w_0^* and w_1^* in the pre-challenge phase. This is done the same way as in [8]. In this case, the challenger does not know when $\mathcal{A}$ queried on w_0^* and w_1^*. If the challenger knew when $\mathcal{A}$ queried on w_0 and w_1, then he could have responded in the same way as in the post-challenge query phase. But if he does not know, the challenger guesses when it is queried with w_0^* or w_1^*. Formally, let $p(\lambda)$ be a bound on the number of queries that adversary $\mathcal{A}$ performs. Also, let the challenger chooses $t_0, t_1 \leftarrow \{0, \cdots, p(\lambda)\}$ in the setup phase. During the pre-challenge phase, if the challenger is queried for an encoding $\mathsf{Encod}_K(w, l)$ of a word w that is the t_0-th or t_1-th distinct word so far, then he acts as if it was queried on w_0^* or w_1^*, respectively, and returns the encoded-word $(g_1^{am_0})^{f_l}$ or $(g_1^{am_1})^{f_l}$ to the adversary $\mathcal{A}$, respectively. Then, in the challenge phase, if it turns out that the guess was wrong, or if the challenger was queried on less than $\max\{t_0, t_1\}$ distinct words, then the challenger aborts and outputs 0. Since the view of adversary $\mathcal{A}$ is independent of the sampling of t_0 and t_1, it holds that the guess of the challenger succeeds with probability of exactly $\frac{1}{(p(\lambda)+1)^2}$, i.e., the success probability is independent of the behavior of $\mathcal{A}$. Consequently, it holds that

3. This tool is available at: https://github.com/generic-group-analyzer/gga (Accessed August 2019).

$$|\Pr[\mathsf{Exp}_{\$\mathsf{F},\mathcal{A}}(\lambda,0)=1]-\Pr[\mathsf{Exp}_{\$\mathsf{F},\mathcal{A}}(\lambda,1)=1]| \quad (3)$$

$$=(p(\lambda)+1)^2\cdot|\Pr[\mathsf{Chal}^{\mathcal{A}}(X_\lambda)=1]-\Pr[\mathsf{Chal}^{\mathcal{A}}(Y_\lambda)=1]|\,. \quad (4)$$

For any PPT adversary $\mathcal{A}$, the bound $p(\lambda)$ on its number of queries is polynomial in the security parameter λ. The MXDHV assumption then implies that the expression in Equation (4) is negligible, and therefore also the expression in Equation (3) is negligible as well, completing the proof of the theorem. □

6 Performance Analysis

In this section, we provide detailed efficiency comparisons between the M-Adjoin constructions of [8] and our proposed constructions: i.e., the non-monotonous scheme of Section 5.1 and the k-monotonous scheme of Section 5.2.

The two constructions presented in [8] satisfy the same security notion, `3Partition`, under two different assumptions in bilinear groups. The security of the first construction is proved under the decision linear assumption [16] and requires four group elements for encoding and four bilinear maps for the adjustment operation. The second construction presented to improve performance, and its security is proved under the seemingly stronger matrix-DDH assumption [21]. The second construction requires two group elements for encoding and two bilinear maps for the adjustment operation.

Table 5 compares all schemes in terms of the underlying hardness assumption, the time complexity of the Encod, Token, Adjust and Eval algorithms (for t columns of length n and a join query of length m in terms of group operations), storage complexity (encoded word size and token size both in terms of group elements), and the achieved security level.

The assumptions in [8] are based on *symmetric* bilinear maps whereas our assumption is based on *asymmetric* bilinear maps. For achieving the same security level, the symmetric bilinear maps require much longer group sizes than the asymmetric ones (3072 bits versus 512 bits for 128-bit security level [22], [23]). This difference makes a significant difference in the computation and storage overheads. Rouselakis and Waters [24] have shown that the assumptions and the security proofs in a symmetric bilinear setting can be translated to an asymmetric bilinear setting in a generic way. Therefore, to make the comparisons fair, we consider the most efficient construction proposed in [8], i.e. M-Adjoin II, with asymmetric pairing and compare our constructions with it.

We have implemented asymmetric M-Adjoin II and our k-monotonous M-Adjoin construction in Java on an Ubuntu 17.04 desktop PC with an Intel Processor 2.9 GHz. We have used Type-F curves[4] for the asymmetric pairing setting of the JPBC library [25]. For our performance analysis, we use the recommended parameters as listed in [22], [23] for the 128-bit security level.

Remark 13. *Since our goal of performance analysis is to compare existing M-Adjoin constructions with the security defined in Section 4 (and not for use in large-scale applications due to the high cost of bilinear map operations in these constructions), we consider our experiment in a small-scale scenario. To this end, we consider columns with sizes 10 to 50, join queries with length 2 to 10 and security levels* $\mathtt{M3P}_1$ *to* $\mathtt{M3P}_5$*. Also, since the* Adjust *and* Eval *algorithms, for computing join result, are applied to all elements of the queried columns and the distribution of these elements does not matter, we randomly initialize the elements of each column.*

4. Type-F curves are commonly used for the asymmetric pairing settings [22], [23].

The output size and execution time of the encoding algorithms over 10 to 50 words are illustrated in Fig. 1a and 1b, respectively. As it can be seen, the output size and execution time are approximately linear with the number of words for each column. Also, the achieved gains are approximately 50% for the output size, and approximately 65% for the execution time.

The output size and execution time of the token generation algorithms over 2 to 10 columns are showed in Fig. 1c and 1d, respectively. As we expected, the output size and execution time are approximately linear with the number of columns for the join query. As it can be seen, our M-Adjoin reduces the storage and computation overheads by approximately 50% and 48%, respectively, compared to asymmetric M-Adjoin II.

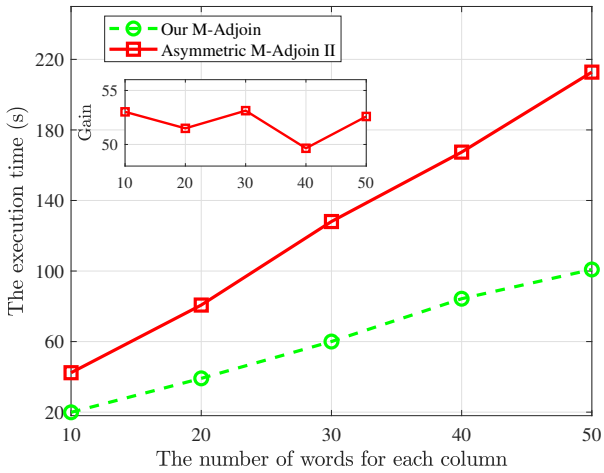


Fig. 2: The join time for achieving $\mathtt{M3P}_1$ security level.

Fig. 2 shows the join time over 10 to 50 words for achieving $\mathtt{M3P}_1$ security level. As it can be seen, the achieved gain is approximately 48% better.

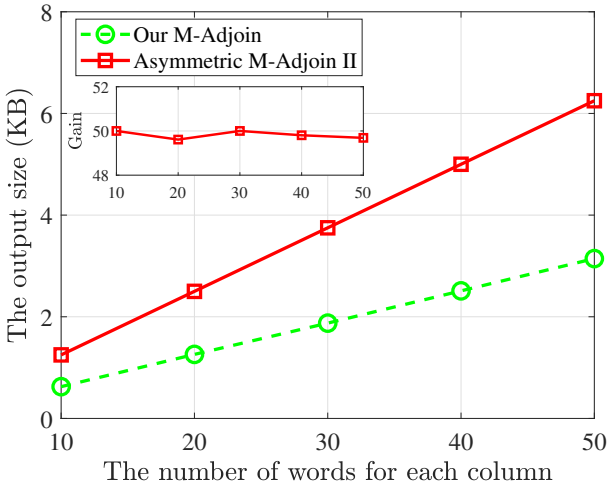
Finally, we consider Table 6 to illustrate the effect of security level and query length on performance. To this end, we consider the columns of size 50, the join queries of length 4 and 6, and the security levels $\mathtt{M3P}_1$ to $\mathtt{M3P}_5$. Generally, performance is expected to decrease at a certain rate as security level increases. But for the settings mentioned, we see that we have less execution time for the security levels $\mathtt{M3P}_2$ and $\mathtt{M3P}_3$ than level $\mathtt{M3P}_1$. In the following, we discuss the reasons for this reduction.

Recall that two algorithms Adjust and Eval are used to compute a join query. According to the required security level ($\mathtt{M3P}_k$), column size (n) and query length (m), Algorithm Adjust is invoked $\left(\left\lceil\frac{(m-1)}{k}\right\rceil(k+1)(n)\right)$ times and Algorithm Eval is invoked $\left(\left\lceil\frac{(m-1)}{k}\right\rceil(n^k)\right)$ times. Therefore,

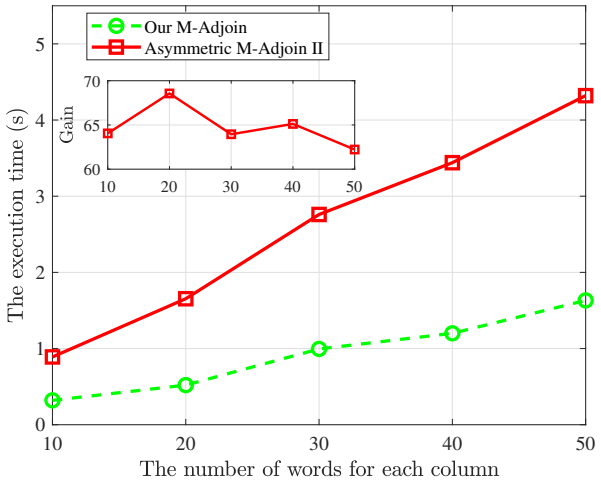
TABLE 5: Asymptotic comparison.

Construction	Assumption	Encod	Token	Adjust	Eval	(Encoded word, Token) size (group elements)	Security
Our non-monotonous	MXDHV	(tn) exp	(m) exp	(mn) bm	(n^{m-1}) mul	(tn, m)	$\mathtt{M3P}_\infty$
Our k-monotonous	MXDHV	(tn) exp	(m) exp	$\left(\left\lceil\frac{(m-1)}{k}\right\rceil(k+1)(n)\right)$ bm	$\left(\left\lceil\frac{(m-1)}{k}\right\rceil(n^k)\right)$ mul	(tn, m)	$\mathtt{M3P}_k$
M-Adjoin II [8]	matix-DDH [21]	$(2tn)$ exp	$(2m)$ exp	$(2n(m-1))$ bm	$(2n(m-1))$ mul	$(2tn, 2m)$	$\mathtt{M3P}_1$
M-Adjoin I [8]	Decision Linear [16]	$(4tn)$ exp	$(4m)$ exp	$(4n(m-1))$ bm	$(4n(m-1))$ mul	$(4tn, 4m)$	$\mathtt{M3P}_1$

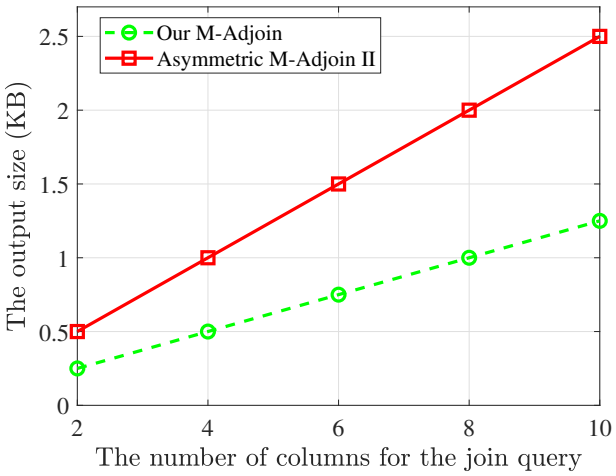
t: the total number of the columns in the database, n: the maximum size of each column in the database, m: the maximum length of each join query, exp: group exponentiation, bm: bilinear map, mul: group multiplication.



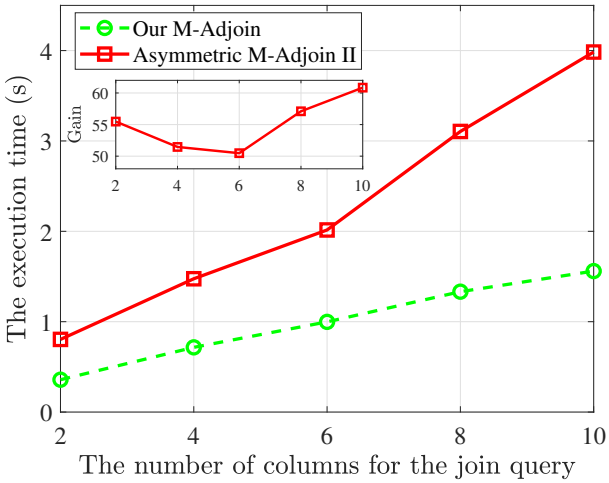
(a) The output size of the Encod algorithm.



(b) The execution time of the Encod algorithm.



(c) The output size of the Token algorithm.



(d) The execution time of the Token algorithm.

Fig. 1: Experimental results for the Token and Encod algorithms.

the reason for reducing the execution time of the security levels $\mathtt{M3P}_2$ and $\mathtt{M3P}_3$ relative to $\mathtt{M3P}_1$ is that the execution time of algorithm Adjust for all three levels is dominant over the execution time of algorithm Eval. However, for the rest of the security levels, Algorithm Eval has more execution time. It should be noted that for large size columns, the join time is always increased when the security level is raised.

Table 6 also shows that, for the considered settings, the performance of our constructions is higher despite the higher security levels, i.e. $\mathtt{M3P}_2$, $\mathtt{M3P}_3$, compared to the asymmetric M-Adjoin II with the security level $\mathtt{M3P}_1$.

The reason for the decrease of overheads is the difference in the employed assumptions. Our constructions use one group element for encoding and one bilinear map for the adjustment operation whereas the asymmetric M-Adjoin II uses two group elements for encoding and two bilinear

TABLE 6: The join time for achieving various security levels.

Construction	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
Our k-monotonous	306	307.5	241.5		
Asymmetric M-Adjoin II	630	–	–	–	–

(a) The join time for 4 columns of length 50.

Construction	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
Our k-monotonous	510	461.25	483	4260	94056
Asymmetric M-Adjoin II	1050	–	–	–	–

(b) The join time for 6 columns of length 50.

maps for the adjustment operation.

7 Conclusions and Future Works

In this paper, we first introduced the syntax and security notion of the multi-adjustable join scheme as a symmetric-key primitive that enables a user to securely outsource his database and to privately issue his join queries on it. We also proposed the `M3Partition` and $\texttt{M3P}_k$ security notions and studied their hierarchical relations. Additionally, we proposed a main scheme that achieves `M3Partition` security but requires $\mathcal{O}(n^{m-1})$ time for joining m columns, each of length n. It remains open if there is a way to get round of this exponential time complexity. On the other hand, our modified scheme, which is only $\texttt{M3P}_k$-secure, with join time $\mathcal{O}\big((m-k)n^k/k\big)$, is quite efficient and might merit to be used in real applications, especially for $k = 1$. But the paid price is a larger leakage.

The future contributions can be considered in several areas such as extending the M-Adjoin schemes to the multi-user models, supporting dynamic storage mechanism and developing the M-Adjoin schemes to the case in which the server is malicious.

References

[1] R. A. Popa, "Building practical systems that compute on encrypted data," Ph.D. dissertation, Massachusetts Institute of Technology, Department of Electrical Engineering , 2014.

[2] 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*. ACM, 2011, pp. 85–100.

[3] R. A. Popa, C. Redfield, and N. Zeldovich, "Cryptdb: processing queries on an encrypted database," vol. 55, no. 9, pp. 103–111, 2012.

[4] R. A. Popa and N. Zeldovich, "Cryptographic treatment of cryptdb's adjustable join," 2012.

[5] J. Furukawa and T. Isshiki, "Controlled joining on encrypted relational database," in *International Conference on Pairing-Based Cryptography*. Springer, 2012, pp. 46–64.

[6] I. Hang, F. Kerschbaum, and E. Damiani, "Enki: access control for encrypted query processing," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 183–196.

[7] S. Kamara and T. Moataz, "Sql on structurally-encrypted databases," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2018, pp. 149–180.

[8] I. Mironov, G. Segev, and I. Shahaf, "Strengthening the security of encrypted databases: non-transitive joins," in *Theory of Cryptography Conference*. Springer, 2017, pp. 631–661.

[9] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," *Journal of Computer Security*, vol. 19, no. 5, pp. 895–934, 2011.

[10] W. Stanek, *Microsoft SQL Server 2008 Administrator's Pocket Consultant: MS SQL Server 2008 Adm PC_p2*. Microsoft Press, 2010.

[11] N. Li, Y. Liu, Y. Dong, and J. Gu, "Application of ant colony optimization algorithm to multi-join query optimization," in *International Symposium on Intelligence Computation and Applications*. Springer, 2008, pp. 189–197.

[12] 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*. Springer, 2013, pp. 353–373.

[13] A. Abadi, S. Terzis, and C. Dong, "Vd-psi: Verifiable delegated private set intersection on outsourced private datasets," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 149–168.

[14] C. C. Aggarwal and S. Y. Philip, *Privacy-preserving data mining: models and algorithms*. Springer Science & Business Media, 2008.

[15] L. Ballard, M. Green, B. De Medeiros, and F. Monrose, "Correlation-resistant storage via keyword-searchable encryption." *IACR Cryptology ePrint Archive*, vol. 2005, p. 417, 2005.

[16] D. Boneh, X. Boyen, and H. Shacham, "Short group signatures," in *Annual International Cryptology Conference*. Springer, 2004, pp. 41–55.

[17] S. D. Galbraith and V. Rotger, "Easy decision diffie-hellman groups," *LMS Journal of Computation and Mathematics*, vol. 7, pp. 201–218, 2004.

[18] M. Scott, "Authenticated id-based key exchange and remote log-in with simple token and pin number." *IACR Cryptology ePrint Archive*, vol. 2002, p. 164, 2002.

[19] V. Shoup, "Lower bounds for discrete logarithms and related problems," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1997, pp. 256–266.

[20] G. Barthe, E. Fagerholm, D. Fiore, J. Mitchell, A. Scedrov, and B. Schmidt, "Automated analysis of cryptographic assumptions in generic group models," in *Annual Cryptology Conference*. Springer, 2014, pp. 95–112.

[21] A. Escala, G. Herold, E. Kiltz, C. Rafols, and J. Villar, "An algebraic framework for diffie–hellman assumptions," *Journal of cryptology*, vol. 30, no. 1, pp. 242–288, 2017.

[22] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, "Recommendation for key management part 1: General (revision 3)," *NIST special publication*, vol. 800, no. 57, pp. 1–147, 2012.

[23] I. ECRYPT, "Yearly report on algorithms and key lengths (2010)," 2011.

[24] Y. Rouselakis and B. Waters, "Practical constructions and new proof methods for large universe attribute-based encryption," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 463–474.

[25] A. De Caro and V. Iovino, "jpbc: Java pairing based cryptography," in *Proceedings of the 16th IEEE Symposium on Computers and Communications, ISCC 2011*, Kerkyra, Corfu, Greece, June 28 - July 1, 2011, pp. 850–855.

Shahram Khazaei is an assistant professor at the Department of Mathematical Sciences at Sharif University of Technology, Iran, since 2012. He received his Ph.D. in computer science from EPFL, Switzerland, in 2010 and was a postdoctoral researcher at KTH Royal Institute of Technology, Sweden, from 2011 to 2012. His main research interests is theoretical and practical aspects of cryptography.

Mojtaba Rafiee obtained his M.Sc. degree in computer science from Shahid Beheshti University, Tehran, Iran, in 2014. He is currently a Ph.D. student of computer science at Sharif University of Technology. His research interest areas include applied cryptography, database security and forensics, verifiable and secure computation.