Multi-Adjustable Join Scheme*

Shahram Khazaei and Mojtaba Rafiee

Sharif University of Technology, Tehran, Iran
shahram.khazaei@sharif.ir,
mojtaba.rafiee67@student.sharif.ir

August 28, 2019

Abstract. In this paper, we introduce the syntax and security notions of multi-adjustable join (M-Adjoin) schemes as an extension of the adjustable join (Adjoin) schemes [Popa-Zeldovich 2012]. An M-Adjoin is a symmetric-key primitive that enables a user to securely outsource his database to a server, and later to privately issue the join queries for a list of column labels (instead of a pair in an Adjoin scheme). The security definitions of Adjoin [Mironov-Segev-Shahaf 2017] extends to M-Adjoin in a straightforward way. It turns out that even though the `3Partition` security does capture the minimal leakage of an Adjoin scheme, it does not carry the expected minimal leakage of an M-Adjoin scheme. We propose a new security notion for this purpose, which we refer to as `M3Partition`. 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 $\mathcal{O}(n^{m-1})$ with minimal leakage. 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 the k-monotonous leakage. It remains open if this barrier is inherent to the security definitions. Our schemes are substantially more efficient (both in computation and storage) than the previous ones. Additionally, we present some separation results between different security definitions, which were left open in previous works.

Keywords: Database outsourcing, Join query, Monotonicity, Non-tranisitivity, Secure outsourcing

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

* Updates of this submission will be available at: `https://eprint.iacr.org/2019/562`.

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. [21,22,23,24], 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 the encrypted databases. Several research such as [15,17,19] 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 list 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 et al. [24], is a symmetric-key primitive that supports the secure join queries on an encrypted database. Recently, Mironov et al. [20] 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. 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 [11] 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 [20] (i.e., the positions in each column with identical contents for every column in the database).

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 stronger than the non-adaptive variant with respect to the minimal leakage. 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. The $\texttt{M3P}_k$ security positions between the non-adaptive and adaptive securities with respect to some leakage function that we refer to as the *k-monotonous leakage*. We call a multi-adjoin 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. This property induces some leakage which, if minimized, is what we call the k-monotonous leakage. That is, in addition to the above-mentioned information revealed by the minimal leakage, the k-monotonous leakage reveal the result pattern for every subset of size $k+1$ of a join query.

Two M-Adjoin schemes with 1-monotonous leakage function have been proposed in [20]. For every integer k, we propose a more flexible and substantially more efficient scheme with k-monotonous leakage function.

The size of adjustment token of our scheme 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 10 for detailed performance comparison and discussion.

A k-monotonous scheme makes it possible to compute the join of $m \geq k+1$ columns, each of length n, in time $\mathcal{O}((m-1)n^k/k)$. Additionally, we propose another construction which is M3Partition-secure (and hence non-monotonous), but it requires $\mathcal{O}(n^{m-1})$ join time.

As an additional contribution, we prove separation results between some security notion that were left open in [20].

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 syntax and security definitions of adjoin are reviewed but they are adopted for the case of multi-adjoin schemes. Section 5 introduces the M3Partition and $\texttt{M3P}_k$ security notions and Section 6 studies how they are positioned in an hierarchy of security levels. Section 7 studies the relations between different leakage-based security definitions and Section 8 summarizes the relations between all security notions. Our two proposed constructions for M-Adjoin, and their security proofs are presented in Section 9. The performance analysis for different M-Adjoin schemes is presented in Section 10. Finally, Section 11 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.

2.3 Basic primitives

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.

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 $Param = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, q, e)$.

[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.

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 3.1 (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*

[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 [23,20] and consider two separate algorithms.

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

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 adapt the security definitions for the Adjoin [20] to M-Adjoin. These definitions can be classified in three categories: 1) the `3Partition` security notion, 2) the indistinguishability-based security notions and 3) the simulation-based security notions. Each of these notions are first explained informally and then the formal definitions are provided. We need the notion of leakage to define the latter two, which will be given in Section 4.2.

4.1 The `3Partition` security notion

The adversary of the `3Partition` notion of security 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 4.1 (`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*

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

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

1. ***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* $\mathcal{A}$*.*
2. ***Pre-challenge query phase:*** *The adversary* $\mathcal{A}$ *may adaptively issue* Addlbl*,* Encod *and* Token *queries, which are defined as follows:*
   (a) $\mathsf{Addlbl}(l, X)$***:*** *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).*
   (b) $\mathsf{Encod}(w, l)$***:*** *computes and returns an encoded-word* $\widetilde{w} \leftarrow \mathsf{Encod}_K(w, l)$ *to the adversary* $\mathcal{A}$*, where* $l \in \mathscr{L} \cup \mathscr{M} \cup \mathscr{R}$*.*
   (c) $\mathsf{Token}(l_1, \cdots, l_m)$***:*** *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* $l_1, \cdots, l_m \in \mathscr{M} \cup \mathscr{R}$*.*
3. ***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* $l \in \mathscr{R}$*.*
4. ***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* $\widetilde{w} \leftarrow \mathsf{Encod}_K(w_b^*, l)$*.*
5. ***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 leakage function

The indistinguishability-based and simulation-based security notions are parameterized with respect to an auxiliary deterministic polynomial-times function Leak, called the *leakage function*. The leakage function models the information that is reveled when a database is outsourced and later queried. The leaked information, called *leakage profile*, typically includes information about the database dimension (DIM), search pattern (SP), duplicate pattern (DP) and result pattern (RP).

A typical leakage function, studied in the literature, is the *minimal leakage function*. In this paper, we consider another one called the *monotonous leakage function*. We present some notations before formally introducing them.

Database and join query. A database is a list of pairs $\mathcal{DB} = \big((\ell_i, C_i)\big)_{i \in [n]}$, where n is the number of columns, $\mathcal{L} = (\ell_i)_{i \in [n]}$ is a list of distinct column labels with $\ell_i \in \{0,1\}^\lambda$ and $C_i = (w_1^i, \cdots, w_{n_i}^i) \in \big(\{0,1\}^\lambda\big)^{n_i}$, is the column with label ℓ_i. It is assumed that the total size of the database, i.e., $N = n + \sum_{i \in [n]} n_i$, is polynomial in security parameter λ. A join query is essentially a list $q = (l_1, \cdots, l_m) \in \mathcal{L}^m$ of column labels, for some integer $m \geq 2$.

Extended algorithms. We extend the Token algorithm such that it outputs $\mathsf{Token}_K(\mathcal{Q}) = \big(\mathsf{Token}_K(q)\big)_{q\in\mathcal{Q}}$ on a list $\mathcal{Q}$ of queries. Similarly, we extend the encoding algorithm to take a column $C = (w_1, \cdots, w_t)$ and a label l, and compute an encoded column $\widetilde{C} \leftarrow \mathsf{Encod}_K(C, l)$ where $\widetilde{C} = (\widetilde{w_1}, \cdots, \widetilde{w_t})$ and $\widetilde{w_i} \leftarrow \mathsf{Encod}_K(w_i, l)$, for $i = 1, \cdots, t$.

Finally, we allow the encoding algorithm to take a key K and a database $\mathcal{DB} = \big((\ell_i, C_i)\big)_{i\in[n]}$ and output an encoded database $\widetilde{\mathcal{DB}} \leftarrow \mathsf{Encod}_K(\mathcal{DB})$ where $\widetilde{\mathcal{DB}} = \big((i, \widetilde{C}_i)\big)_{i\in[n]}$ where $\widetilde{C}_i = \mathsf{Encod}_K(C_i, \ell_i)$.

Equality pattern and search pattern. We define the equality pattern and search pattern of a join query $q = (\ell_{i_1}, \cdots, \ell_{i_m})$, respectively, as follows

$$\mathrm{EQ}(q) = \mathrm{EQ}(\ell_{i_1}, \cdots, \ell_{i_m}) = \big((k_1, \cdots, k_m) \mid \ w_{k_1}^{i_1} = \cdots = w_{k_m}^{i_m}\big) \ ,$$

and

$$\mathrm{I}(q) = (i_1, \ldots, i_m) \ .$$

Below, we formally define the minimal and k-monotonous leakage functions, where k is an integer parameter. The intuition behind our monotonous leakage will be clear in Section 5.

Definition 4.2 (leakage functions) *For a database $\mathcal{DB} = \big((\ell_i, C_i)\big)_{i\in[n]}$, a list of join queries $\mathcal{Q}$ and an integer k, we define the minimal leakage and the k-monotonous leakage function, respectively, as follows:*

$$(SP, DIM, DP, RP) \leftarrow \mathsf{MinLeak}(\mathcal{DB}, \mathcal{Q}) \ ,$$
$$(SP, DIM, DP, RP, MP_k) \leftarrow \mathsf{MonLeak}_k(\mathcal{DB}, \mathcal{Q}) \ ,$$

where the leakage profile includes:

- **Search pattern.** $SP = \mathcal{I} = \big(\mathrm{I}(q)\big)_{q\in\mathcal{Q}}$ *is the search pattern,*
- **Dimension.** $DIM = (|C_i|)_{i\in[n]}$ *is the database dimensions,*
- **Duplication pattern.** $DP = \Big(\mathrm{EQ}\big(\mathrm{I}^{-1}(i, i)\big)\Big)_{i\in[n]}$ *is the duplication pattern,*
- **Result pattern.** $RP = \big(\mathrm{EQ}(\mathrm{I}^{-1}(x))\big)_{x\in\mathcal{I}}$ *is the result pattern,*
- **Monotonicity pattern.** $MP_k = \big(\mathrm{EQ}(\mathrm{I}^{-1}(x'))\big)_{x'\in\mathcal{I}_k}$ *is the monotonicity pattern, where $\mathcal{I}_k$ includes every subset of size $k+1$ of every indexed query; that is, $\mathcal{I}_k = (y \ : \ y \subseteq x, |y| = k+1, x \in \mathcal{I})$.*

Remark 4.3 (Label-hiding v.s. label-leaking profile) *Our leakage functions do not reveal the column labels. One may also consider a* label-leaking *variants of the above leakage functions where the label set $\mathcal{L}$ is also included in the leakage profile. Hiding labels, however, is not a big burden in practice since it can be handled using standard techniques; e.g., see [20, Section 4.4]. Nevertheless, as we will see in Section 7 (Proposition 7.5), the label-hiding property is crucial for proving separation between adaptive simulation-based security and adaptive indistinguishability-based security definitions, which was left open in [20].*

4.3 Indistinguishability-based security notions

The indistinguishability-based security notion comes in two flavours, depending on the adversary which may be adaptive or non-adaptive. The definitions can be given for a general leakage function. But to keep our discussion simple, we assume that it is the minimal leakage function.

Non-adaptive IND security. In this notion of security, the adversary first receives the public parameters of an M-Adjoin scheme. It then chooses a pair of challenge databases $\mathcal{DB}_0, \mathcal{DB}_1$, with the same column labels[3] and a challenge query list $\mathcal{Q}$ such that $\mathsf{Leak}(\mathcal{DB}_0, \mathcal{Q}) = \mathsf{Leak}(\mathcal{DB}_1, \mathcal{Q})$. Such an adversary is called a valid adversary.

The non-adaptive indistinguishability-based notion of security requires that such an adversary should not be able to distinguish between $\big(\mathsf{Encod}_K(\mathcal{DB}_0), \mathsf{Token}_K(\mathcal{Q})\big)$ and $\big(\mathsf{Encod}_K(\mathcal{DB}_1), \mathsf{Token}_K(\mathcal{Q})\big)$ with a non-negligible advantage. The formal definition follows.

Definition 4.4 (Non-adaptive IND security) *An M-Adjoin join scheme such as $\Pi = (\mathsf{Gen}, \mathsf{Encod}, \mathsf{Token}, \mathsf{Adjust}, \mathsf{Eval})$ is called non-adaptively indistinguishable-based secure with respect to the leakage function* Leak *(abbreviated* Leak-*naIND*-*secure) against non-adaptive adversaries if for all PPT valid adversary $\mathcal{A}$, there exists a negligible function ε such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{naIND}}_{\Pi,\mathcal{A}}(\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{naIND}}_{\Pi,\mathcal{A}}(\lambda, 0) = 1] - \\ \Pr[\mathsf{Exp}^{\mathsf{naIND}}_{\Pi,\mathcal{A}}(\lambda, 1) = 1]| \leq \varepsilon(\lambda),\end{aligned}$$

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

1. ***Setup phase:*** *The challenger* Chal *samples $(Param, K) \leftarrow \mathsf{Gen}(1^\lambda)$ and sends the public parameters $Param$ to the adversary $\mathcal{A}$.*
2. ***Challenge phase:*** *The adversary $\mathcal{A}$ chooses two databases $\mathcal{DB}_0$, $\mathcal{DB}_1$, with the same column labels, and a list of queries $\mathcal{Q}$ such that $\mathsf{Leak}(\mathcal{DB}_0, \mathcal{Q}) = \mathsf{Leak}(\mathcal{DB}_1, \mathcal{Q})$. As a response, the adversary $\mathcal{A}$ obtains $\mathsf{Encod}_K(\mathcal{DB}_b)$ and $\mathsf{Token}_K(\mathcal{Q})$.*
3. ***Output phase:*** *The adversary $\mathcal{A}$ outputs a value $\sigma \in \{0, 1\}$ which is defined as the output of the experiment.*

[3] This requirement has not been explicitly stated in [20, Definition 4.2]. However, without this requirement, the definition becomes useless since the `aIND` does not imply the `naIND` security if the leakage function is label-hiding; see Remark 4.3. For a label-leaking leakage function, this requirement is redundant, as the label set is already included in the leakage profile.

Adaptive IND security. The adversary of the adaptive indistinguishability-based notion of security, first receives the public parameters of the M-Adjoin scheme. It can then adaptively issue a pair of words w_0, w_1 and a column label l, and obtain an encoding $\mathsf{Encod}_K(w_b, l)$, for some fixed bit $b \in \{0,1\}$. In this type of query, the adversary gradually constructs two databases $\mathcal{DB}_0, \mathcal{DB}_1$ with the same dimension and the same column label set. The adversary can adaptively issue a list of column labels, and obtain a token for computing their join. Let $\mathcal{Q}$ be a list of queries that the adversary has obtained their tokens at a given point of time. Such an adversary is called valid if after having made each query it holds that $\mathsf{Leak}(\mathcal{DB}_0, \mathcal{Q}) = \mathsf{Leak}(\mathcal{DB}_1, \mathcal{Q})$.

The adaptive indistinguishability-based notion of security asks that such an adversary should not be able to distinguish between the game in which $b = 0$ and the game with $b = 1$, with a non-negligible advantage. The formal definition is given below.

In order to be precise, we present the following definition, before giving the formal security definition.

Definition 4.5 (Inserting into a database) *Let $\mathcal{DB} = \big((\ell_i, C_i)\big)_{i\in[n]}$ be a database and $(w, l) \in \{0,1\}^\lambda \times \{0,1\}^\lambda$ be a word/label pair. By insertion of (w, l) into $\mathcal{DB}$ we get a new database as follows. If l already exists in the column label set, i.e., $l = \ell_i$ for some $i \in [n]$, then w is appended to the end of C_i; otherwise, a single-word column $C_{n+1} = (w)$ with label $\ell_{n+1} = l$, i.e., the pair $(l, (w))$ is appended to the database.*

Definition 4.6 (Adaptive IND security) *An M-Adjoin scheme such as $\Pi = (\mathsf{Gen}, \mathsf{Encod}, \mathsf{Token}, \mathsf{Adjust}, \mathsf{Eval})$ is called adaptively indistinguishable-based secure with respect to the leakage function Leak (abbreviated Leak-a*IND*-secure) against adaptive adversaries if for all PPT valid adversary $\mathcal{A}$, there exists a negligible function ε such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{aIND}}_{\Pi,\mathcal{A}}(\lambda) = |\Pr[\mathsf{Exp}^{\mathsf{aIND}}_{\Pi,\mathcal{A}}(\lambda, 0) = 1] - \\ \Pr[\mathsf{Exp}^{\mathsf{aIND}}_{\Pi,\mathcal{A}}(\lambda, 1) = 1]| \leq \varepsilon(\lambda),\end{aligned}$$

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

1. ***Setup phase:*** *The challenger* Chal *samples $(Param, K) \leftarrow \mathsf{Gen}(1^\lambda)$ and sends the public parameters $Param$ to the adversary $\mathcal{A}$. In addition, two empty databases $\mathcal{DB}_0, \mathcal{DB}_1$ and an empty list $\mathcal{Q}$ for the queries are initialized.*
2. ***Challenge phase:*** *The adversary $\mathcal{A}$ may adaptively issue $\mathsf{Encod}'_K(\cdot,\cdot,\cdot)$ and $\mathsf{Token}_K(\cdot)$ queries, which are defined as follows:*
   - *$\mathsf{Encod}'_K(w_0, , w_1, l)$: The pairs (w_0, l) and (w_1, l) are inserted into the database $\mathcal{DB}_0$ and $\mathcal{DB}_1$, respectively. Then, the challenger sends an encoded-word $\mathsf{Encod}_K(w_b, l)$ to the adversary $\mathcal{A}$.*
   - *$\mathsf{Token}_K(q)$: The query $q = (l_1, \cdots, l_m)$ is inserted into the list $\mathcal{Q}$, and the challenger sends the adjustments tokens $(at_1, \cdots, at_m) \leftarrow \mathsf{Token}_K(q)$ to the adversary $\mathcal{A}$.*

It is required that, after issuing each query, the equality $\mathsf{Leak}(\mathcal{DB}_0, \mathcal{Q}) = \mathsf{Leak}(\mathcal{DB}_1, \mathcal{Q})$ *holds.*

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

4.4 Simulation-based security notions

The simulation-based security notion also has the adaptive and non-adaptive variants. Again, for simplicity we assume that the leakage function is the minimal one.

Non-adaptive SIM security. The non-adaptive simulation-based notion of security considers an adversary, a simulator and two worlds called real and ideal.

In the real-world, the adversary interacts with the M-Adjoin scheme and receives the public parameters of the M-Adjoin scheme at the beginning. Then, it chooses a database $\mathcal{DB}$ and a list $\mathcal{Q}$ of queries and obtains an encoded database and a list of join tokens for each query $q \in \mathcal{Q}$.

In the ideal-world, the adversary, however, interacts with a simulator. It gets the public parameters produced by the simulator and chooses a database $\mathcal{DB}$ and a list $\mathcal{Q}$ of queries. It then obtains an encoded database and a list of join tokens for each $q \in \mathcal{Q}$, produced by the simulator using the leakage profile $\mathsf{Leak}(\mathcal{DB}, \mathcal{Q})$, without having access to the key.

The non-adaptive simulation-based notion of security requires that such an adversary should not be able to distinguish whether he interacts with the M-Adjoin scheme or with the simulator, unless with a negligible advantage. The formal definition is given below.

Definition 4.7 (Non-adaptive SIM security) *An M-Adjoin scheme such as* $\Pi = (\mathsf{Gen}, \mathsf{Encod}, \mathsf{Token}, \mathsf{Adjust}, \mathsf{Eval})$ *is said to be non-adaptively simulation-based secure with respect to the leakage function* Leak *(abbreviated* Leak-*naSIM-secure) against non-adaptive adversaries if for every PPT adversary* $\mathcal{A}$ *there exist a PPT simulator* $\mathcal{S}$ *and a negligible function* ε *such that*

$$|\Pr[\mathsf{Exp}^{\mathsf{Real,naSIM}}_{\Pi,\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{Ideal,naSIM}}_{\Pi,\mathcal{A},\mathcal{S}}(\lambda) = 1]| \leq \varepsilon(\lambda),$$

where the experiments $\mathsf{Exp}^{\mathsf{Real,naSIM}}_{\Pi,\mathcal{A}}(\lambda)$ *and* $\mathsf{Exp}^{\mathsf{Ideal,naSIM}}_{\Pi,\mathcal{A},\mathcal{S}}(\lambda)$ *are defined as follows.*

Real world $(\mathsf{Exp}^{\mathsf{Real,naSIM}}_{\Pi,\mathcal{A}}(\lambda))$**:**

1. ***Setup phase:*** *The challenger samples* $(Param, K) \leftarrow \mathsf{Gen}(1^\lambda)$*, and returns the public parameters* $Param$ *to the adversary* $\mathcal{A}$*.*
2. ***Challenge phase:*** *The adversary* $\mathcal{A}$ *takes the security parameter* λ*, chooses a database* $\mathcal{DB}$ *and a list of queries* $\mathcal{Q}$*, and sends them to the challenger.*
3. ***Response phase:*** *The challenger computes* $\widetilde{\mathcal{DB}} \leftarrow \mathsf{Encod}_K(\mathcal{DB})$ *and* $AT \leftarrow \mathsf{Token}_K(\mathcal{Q})$*, and returns them to the adversary* $\mathcal{A}$*.*
4. ***Output phase:*** *The adversary* $\mathcal{A}$ *outputs a value* $\sigma \in \{0, 1\}$ *which is defined as the output of the experiment.*

Ideal world ($\mathsf{Exp}^{\mathsf{Ideal,naSIM}}_{\Pi,\mathcal{A},\mathcal{S}}(\lambda)$)**:**

1. ***Setup phase:*** *The simulator $\mathcal{S}$ produces the public parameters $Param^*$, which are given as input to the adversary $\mathcal{A}$.*
2. ***Challenge phase:*** *The adversary $\mathcal{A}$ takes the security parameter λ, chooses a database $\mathcal{DB}$ and a list of queries $\mathcal{Q}$, and sends them to the challenger.*
3. ***Response phase:*** *The challenger computes the leakage profile $L \leftarrow \mathsf{Leak}(\mathcal{DB}, \mathcal{Q})$, and gives L as input to the simulator. The simulator $\mathcal{S}$ produces an encoded database $\widetilde{\mathcal{DB}}^*$ and a list of tokens AT^*, which are then given to the adversary $\mathcal{A}$.*
4. ***Output phase:*** *The adversary $\mathcal{A}$ outputs a value $\sigma \in \{0,1\}$ which is defined as the output of the experiment.*

Adaptive SIM security. The adaptive simulation-based notion of security also considers an adversary, a simulator and the real and ideal worlds.

In the real-world, the adversary interacts with the M-Adjoin scheme and receives the public parameters of the M-Adjoin scheme. It can then adaptively issue a pair of word/label (w, l), and obtain an encoding $\mathsf{Encod}_K(w, l)$. The adversary can also adaptively issue a query, and obtain a token for computing the join.

In the ideal-world, the adversary, however, interacts with the simulator. It obtains the public parameters produced by the simulator. Then, the adversary can adaptively issue a pair of word/label (w, l), and obtain an encoding $\mathsf{Encod}_K(w, l)$, produced by the simulator using the leakage profile $L \leftarrow \mathsf{Leak}(\mathcal{DB}, \mathcal{Q})$. It can also adaptively issue a query, and obtain a token for computing the join, produced by the simulator using the leakage profile $L \leftarrow \mathsf{Leak}(\mathcal{DB}, \mathcal{Q})$.

The adaptive simulation-based notion of security requires that such an adversary should not be able to distinguish whether it interacts with the M-Adjoin scheme or with the simulator, unless with a negligible advantage. The formal definition follows.

Definition 4.8 (Adaptive SIM security) *An M-Adjoin scheme such as $\Pi = (\mathsf{Gen}, \mathsf{Encod}, \mathsf{Token}, \mathsf{Adjust}, \mathsf{Eval})$ is said to be adaptively simulation-based secure with respect to the leakage function* Leak *(abbreviated* Leak-*aSIM-secure) against adaptive adversaries if for every PPT adversary $\mathcal{A}$ there exist a PPT simulator $\mathcal{S}$ and a negligible function ε such that*

$$|\Pr[\mathsf{Exp}^{\mathsf{Real,aSIM}}_{\Pi,\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{Ideal,aSIM}}_{\Pi,\mathcal{A},\mathcal{S}}(\lambda) = 1]| \leq \varepsilon(\lambda),$$

where the experiments $\mathsf{Exp}^{\mathsf{Real,aSIM}}_{\Pi,\mathcal{A}}(\lambda)$ and $\mathsf{Exp}^{\mathsf{Ideal,aSIM}}_{\Pi,\mathcal{A},\mathcal{S}}(\lambda)$ are defined as follows.

Real world ($\mathsf{Exp}^{\mathsf{Real,aSIM}}_{\Pi,\mathcal{A}}(\lambda)$)**:**

1. ***Setup phase:*** *The challenger samples $(Param, K) \leftarrow \mathsf{Gen}(1^\lambda)$, and returns the public parameters $Param$ to the adversary $\mathcal{A}$.*

2. ***Challenge phase:*** *The adversary $\mathcal{A}$ may adaptively issue* $\mathsf{Encod}_K(\cdot,\cdot)$ *and* $\mathsf{Token}_K(\cdot)$ *queries and receive back the output.*
3. ***Output phase:*** *The adversary $\mathcal{A}$ outputs a value $\sigma \in \{0,1\}$ which is defined as the output of the experiment.*

Ideal world $(\mathsf{Exp}^{\mathsf{Ideal},\mathsf{aSIM}}_{\Pi,\mathcal{A},\mathcal{S}}(\lambda))$**:**

1. ***Setup phase:*** *The simulator $\mathcal{S}$ produces the public parameters $Param^*$, which are given as input to the adversary $\mathcal{A}$. An empty database $\mathcal{DB}$ and an empty list $\mathcal{Q}$ of column labels are initialized.*
2. ***Challenge phase:*** *The adversary $\mathcal{A}$ may adaptively issue* $\mathsf{Encod}(\cdot,\cdot)$ *and* $\mathsf{Token}(\cdot)$ *queries, which are defined as follows:*
   - $\mathsf{Encod}(w,l)$*: The pair (w,l) is inserted into the database $\mathcal{DB}$, and the simulator $\mathcal{S}$ obtains the leakage profile $L \leftarrow \mathsf{Leak}(\mathcal{DB},\mathcal{Q})$. Finally, the simulator $\mathcal{S}$ sends an encoded-word $\widetilde{w}^*$ to the adversary $\mathcal{A}$.*
   - $\mathsf{Token}(q)$*: The query $q = (l_1,\cdots,l_m)$ is inserted into the list $\mathcal{Q}$, and the simulator $\mathcal{S}$ obtains the leakage profile $L \leftarrow \mathsf{Leak}(\mathcal{DB},\mathcal{Q})$. Finally, the simulator $\mathcal{S}$ sends a list of adjustment tokens $(at_1^*,\cdots,at_m^*)$ to the adversary $\mathcal{A}$.*
3. ***Output phase:*** *The adversary $\mathcal{A}$ outputs a value $\sigma \in \{0,1\}$ which is defined as the output of the experiment.*

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

In this section, we introduce a new security notion for M-Adjoin schemes, called `M3Partition`, which stands between the non-adaptive and adaptive securities with respect to the minimal leakage (Definition 4.2). Additionally, we introduce a family $\{\mathtt{M3P}_k\}_{k\in\mathbb{N}}$ of security notions such that $\mathtt{M3P}_k$ positions between the non-adaptive and adaptive securities with respect to the k-monotonous leakage (Definition 4.2). Additionally, $\mathtt{M3P}_k$ lies between $\mathtt{M3P}_{k-1}$ and $\mathtt{M3P}_{k+1}$. Our definition is a natural extension of the other definitions such that `3Partition` corresponds to $\mathtt{M3P}_1$ and one may imagine that `M3Partition` corresponds to $\mathtt{M3P}_\infty$.

5.1 Monotonous M-Adjoin schemes

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 is not sufficient for capturing the minimal leakage function defined. 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 5.1 (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 $(l_i)_{i\in A}$.*

Remark 5.2 (More general monotonicity property) *The k-monotonicity property corresponds to a threshold leakage which is reminiscent of the Shamir-Blakely's [26,7] threshold access structure in secret sharing schemes. Similar to Ito-Saito-Nishizeki's [18] generalized notion of access structures, one can work with more general monotonous properties. It is unclear to us if such a property can be used in an interesting way in real applications.*

The M-Adjoin schemes of [20] have the 1-monotonicity property with the adjustment token size $\mathcal{O}(m)$. For every integer k, we propose a more flexible and substantially more efficient 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.

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 $k+2$. This property induces some leakage which is not captured by the minimal leakage but reflected in the k-monotonous leakage, introduced in Section 4.2. For our convenience, we define a weaker version of the monotonicity property.

Definition 5.3 (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 $A \subseteq [m]$".*

5.2 An illustrative example

Consider an M-Adjoin scheme with the weak k-monotonicity property. We will argue that it cannot be $\mathtt{naIND}$-secure (and more generally $\mathtt{M3P}_{k+1}$-secure, see Lemma 5.7) with respect to the k-monotonous leakage function (Definition 4.2). Without loss of generality, assume that the weak k-monotonicity property holds for the set $A = \{1, \cdots, k+1\}$.

The adversary constructs two databases $\mathcal{DB}_0$ and $\mathcal{DB}_1$, each containing $k+1$ single-word columns with labels $\ell_1, \ldots, \ell_{k+1}$. Denote the column with label ℓ_i in the database $\mathcal{DB}_j$ with C_i^j, where $i \in \{1, \ldots, k+1\}$ and $j \in \{0, 1\}$. The adversary chooses $k+1$ distinct words $w_1, \ldots, w_{k+1}$ and constructs $\mathcal{DB}_0$ and $\mathcal{DB}_1$ as in Table 1. That is, all the columns of the database $\mathcal{DB}_0$ are distinct and hence the intersection of any subset of size k as well as the intersection of all of them is empty. However, in $\mathcal{DB}_1$, the first k columns are identical but distinct from the last one. Therefore, the intersection of the fist k columns is non-empty but the intersection of all of them is empty.

Table 1. Adversary's strategy

Database	$\mathcal{DB}_0$				$\mathcal{DB}_1$			
Column label	ℓ_1	$\cdots$	ℓ_k	ℓ_{k+1}	ℓ_1	$\cdots$	ℓ_k	ℓ_{k+1}
C_i	w_1	$\cdots$	w_k	w_{k+1}	w_1	$\cdots$	w_1	w_{k+1}

The adversary chooses a single query $q_1 = (\ell_1, \ldots, \ell_{k+1})$ and sends it to the challenger. Notice that the adversary is valid since $\mathsf{MonLeak}_k(\mathcal{DB}_0, \mathcal{Q}) = \mathsf{MonLeak}_k(\mathcal{DB}_1, \mathcal{Q})$, where $\mathcal{Q} = \{q\}$. The adversary has received an encoding of $\mathcal{DB}_b$, where $b \in \{0, 1\}$ is unknown to him, along with an adjustment token $(at_1, \ldots, at_{k+1})$ for q_1. By the monotonicity property, the adversary has a valid token $(at_1, \ldots, at_k)$ for the unqueried query $q_2 = (\ell_1, \ldots, \ell_k)$, using which he can determine the value of $b \in \{0, 1\}$ since $\mathsf{MonLeak}_k(\mathcal{DB}_0, \mathcal{Q}') \neq \mathsf{MonLeak}_k(\mathcal{DB}_1, \mathcal{Q}')$, where $\mathcal{Q}' = \{q_1, q_2\}$. Then, the adversary outputs $\sigma = b$. Clearly, the advantage of the adversary is 1.

5.3 Informal definitions

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.

`M3P`$_k$ security. For every integer k, we define the `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 `3Partition` experiment can be viewed as the limit of the `M3P`$_k$ experiment when k goes to infinity. Therefore, we use `M3Partition` and `M3P`$_\infty$ interchangeably.

5.4 Formal definition

Below, we provide a formal definition of the `M3Partition` (i.e., $\texttt{M3P}_\infty$) and $\texttt{M3P}_k$ security notions.

Definition 5.4 ($\texttt{M3P}_k$ 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 $\mathit{M3P}_k$-secure if for all PPT algorithms $\mathcal{A}$, there exists a negligible function ε such that*

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

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

1. ***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 $\mathcal{A}$.*
2. ***Pre-challenge query phase:*** *The adversary $\mathcal{A}$ may adaptively issue* Addlbl, Encod *and* Token *queries, which are defined as follows:*
   (a) $\mathsf{Addlbl}(l, X)$***:*** *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).*
   (b) $\mathsf{Encod}(w, l)$***:*** *computes and returns an encoded-word $\widetilde{w} \leftarrow \mathsf{Encod}_K(w, l)$ to the adversary $\mathcal{A}$, where $l \in \mathscr{L} \cup \mathscr{M} \cup \mathscr{R}$.*
   (c) $\mathsf{Token}(l_1, \cdots, l_m)$***:*** *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 $m \leq k+1$.*
3. ***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 $l \in \mathscr{R}$.*
4. ***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 $\widetilde{w} \leftarrow \mathsf{Encod}_K(w_b^*, l)$.*
5. ***Output phase:*** *The adversary $\mathcal{A}$ outputs a value $\sigma \in \{0,1\}$ which is defined as the output of the experiment.*

Definition 5.5 (`M3Partition` security) *An $\mathit{M3P}_\infty$-secure M-Adjoin scheme is simply called `M3Partition`-secure.*

5.5 Security relations

The following corollary trivially follows by security definitions.

Corollary 5.6 (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 5.7 (Weak k-monotonicity $\implies \sim \mathtt{M3P}_{k+1}$) *An M-Adjoin scheme with the 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 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 M3Partition experiment is 1. □

Proposition 5.8 (Separations) *Let k be an integer. Then,*

(a) **($\mathtt{M3P}_k \not\Longrightarrow \mathtt{M3P}_{k+1}$)** *An $\mathtt{M3P}_k$-secure M-Adjoin scheme is not necessarily $\mathtt{M3P}_{k+1}$-secure.*
(b) **($\mathtt{M3P}_k \not\Longrightarrow$ 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 5.7, it is not $\mathtt{M3P}_{k+1}$-secure, proving Part (a). Part (b) follows by Lemma 5.7.

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 $q' = (\ell_1, \ldots, \ell_{k+1})$ 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 subquery $q' = (\ell_1, \ldots, \ell_{k+1})$ when $m \geq k+2$) does not provide anything new to adversary since he was already allowed to issue such a query. □

6 Positions of M3Partition and $\mathtt{M3P}_k$

For the case of Adjoin, the relations between the 3Partition security and simulation/indistinguishability-based security notions with respect to the minimal leakage function have been studied in [20, Claims 4.3 and 4.5]. Their proofs extend to the case of M-Adjoin in a straightforward way, giving rise to the following two propositions.

Proposition 6.1 (Position of M3Partition) *The M3Partition security lies between the adaptive and non-adaptive indistinguishability-based securities with respect to the minimal leakage function. That is,*

(a) (MinLeak-aIND $\implies$ M3Partition) *Any* MinLeak-*aIND-secure M-Adjoin scheme is M3Partition-secure, too.*
(b) (M3Partition $\implies$ MinLeak-naIND) *Any M3Partition-secure scheme is* MinLeak-*naIND-secure, too.*

Proposition 6.2 (Position of $\mathtt{M3P}_k$) *The $\mathtt{M3P}_k$ security lies between the adaptive and non-adaptive indistinguishability-based securities with respect to the k-monotonous leakage function. That is,*

(a) ($\mathsf{MonLeak}_k$-aIND $\implies$ M3Pk) *Any* $\mathsf{MonLeak}_k$-*aIND-secure M-Adjoin scheme is* $\mathtt{M3P}_k$*-secure, too.*
(b) ($\mathtt{M3P}_k$ $\implies$ $\mathsf{MonLeak}_k$-naIND) *Any* $\mathtt{M3P}_k$*-secure scheme is* $\mathsf{MonLeak}_k$-*naIND-secure, too.*

For the case of Adjoin in [20, Page 641, Section 1.4], it has been argued that for databases with a logarithmic number of columns the 3Partition security and aIND-security with minimal leakage are equivalent. The same argument also applies here to show that, for the M-Adjoin schemes, the M3Partition (resp. $\mathtt{M3P}_k$) security is equivalent to the aIND-security with regards to the minimal leakage (resp. k-monotonous minimal leakage). However, it remains open if this is true for databases with a super-logarithmic number of columns.

Additionally proving/refuting equivalence between the 3Partition security and naIND-security (with minimal leakage) was left unexpressed for the case of Adjoin in [20]. The following proposition, which is proved using standard techniques for separating non-adaptive and adaptive security notions, shows that they are indeed inequivalent.

Proposition 6.3 (naIND $\not\Longrightarrow$ 3Partition) *For any leakage function* Leak*, the* Leak*-naIND-security of an M-Adjoin scheme does not necessarily imply its 3Partition security.*

Proof. Let $\Pi = (\mathsf{Gen}, \mathsf{Encod}, \mathsf{Token}, \mathsf{Adjust}, \mathsf{Eval})$ be some naIND-secure M-Adjoin scheme. We construct an M-Adjoin scheme $\widetilde{\Pi} = (\widetilde{\mathsf{Gen}}, \widetilde{\mathsf{Encod}}, \widetilde{\mathsf{Token}}, \widetilde{\mathsf{Adjust}}, \widetilde{\mathsf{Eval}})$ which is yet naIND-secure, with the same leakage function, however it is not 3Partition secure. The scheme $\widetilde{\Pi}$ is as follows:

- $\widetilde{\mathsf{Gen}}(1^\lambda)$**:** It runs $(Param, K) \leftarrow \mathsf{Gen}(1^\lambda)$ and chooses a random word $w^* \in \{0,1\}^\lambda$, and outputs $(Param, \widetilde{K})$, where $\widetilde{K} = (K, w^*)$.
- $\widetilde{\mathsf{Encod}}_{\widetilde{K}}(w, l)$**:** It computes an encoded-word $\widetilde{w}$ as

$$\widetilde{w} = \widetilde{\mathsf{Encod}}_{\widetilde{K}}(w,l) = \begin{cases} (\mathsf{Encod}_K(w,l), w^*) & w \neq w^* \\ w^* & w = w^* \end{cases}.$$

It is easy to see that $\widetilde{\Pi}$ is still naIND-secure without any change in the leakage function, since the adversary's advantage only increases by $\frac{1}{2^\lambda}$ due to his ability to guess w^*. It is also easy to construct a 3Partition-attacker for $\widetilde{\Pi}$ with advantage one, since the adversary learns w^* after issuing the first query. He will then use it as one of his challenge words. □

7 Relations between leakage-based security notions

The reader may recall definitions of the minimal and k-monotonous leakage function (Definition 4.2) before continuing this section. The following "figurative" relation in terms of the amount of leakage is trivial:

$$\mathsf{MinLeak} \leq \mathsf{MonLeak}_{k+1} \leq \mathsf{MonLeak}_k \ ,$$

which can be formally stated by the following corollary.

Corollary 7.1 (Leakage implications) *Let* $X \in \{$*aSIM*, *naSIM*, *aIND*, *naIND*$\}$ *and* k *be an integer.*

(a) ($\mathsf{MinLeak} \implies \mathsf{MonLeak}_k$) *If an M-Adjoin scheme is* $\mathsf{MinLeak}$*-X-secure scheme, it is* $\mathsf{MonLeak}_k$*-X-secure, too.*

(b) ($\mathsf{MonLeak}_{k+1} \implies \mathsf{MonLeak}_k$) *If an M-Adjoin scheme is* $\mathsf{MonLeak}_{k+1}$*-X-secure scheme, it is* $\mathsf{MonLeak}_k$*-X-secure, too.*

The proof of the following proposition is similar to the proof of Proposition 5.8 and is left to the reader.

Proposition 7.2 (Leakage separation) *Let* $X \in \{aSIM, naSIM, aIND, naIND\}$ *and* k *be an integer. Then,*

(a) ($\mathsf{MonLeak}_{k+1} \not\Longrightarrow \mathsf{MonLeak}_k$) *A* $\mathsf{MonLeak}_{k+1}$-X*-secure scheme is not necessarily* $\mathsf{MonLeak}_k$-X*-secure.*
(b) ($\mathsf{MonLeak}_k \not\Longrightarrow \mathsf{MinLeak}$) *A* $\mathsf{MonLeak}_k$-X*-secure scheme is not necessarily* MinLeak-X*-secure.*

The proofs of the following propositions are pretty much similar to the corresponding claims in [20] for the case of Adjoin schemes, and hence omitted here.

Proposition 7.3 (aSIM $\Longrightarrow$ aIND **[20, Claim 4.9]**) *For any leakage function* Leak*, if an M-Adjoin scheme is* Leak-$aSIM$*-secure, then it is* Leak-$aIND$*-secure too.*

Proposition 7.4 (naSIM $\Longrightarrow$ naIND **[20, Claim 4.7]**) *Let* Leak *be any of the leakage functions defined in Definition 4.2 (i.e,* MinLeak *or* $\mathsf{MonLeak}_k$*). If an M-Adjoin scheme is* Leak-$naSIM$*-secure, then it is* Leak-$naIND$*-secure too.*

It is easy to see that Corollary 7.1 and Propositions 7.2, 7.3 and 7.4 all still hold true if the leakage functions are assumed to be *label-leaking* instead of being *label-hiding* as in Definition 4.2; see Remark 4.3.

The following proposition—which was left unanswered for the case of Adjoin in [20]– separates the aIND and aSIM security definitions with respect to leakage functions defined in Definition 4.2. The point that we take advantage of is that we consider the leakage functions to be label-hiding. It remains open if the separation still holds for label-leaking leakage functions.

Proposition 7.5 (aIND $\not\Longrightarrow$ aSIM) *Let* Leak *be any of the leakage functions defined in Definition 4.2 (i.e,* MinLeak *or* $\mathsf{MonLeak}_k$*). Then, the* Leak-$aIND$ *security does not necessarily imply the* Leak-$aSIM$ *security.*

Proof. Let Π be some Leak-aIND-secure M-Adjoin scheme. We construct an M-Adjoin scheme $\widetilde{\Pi}$ which is still Leak-aIND-secure, but it is not Leak-aSIM-secure. All algorithms of $\widetilde{\Pi}$ are the same as Π except the encoding algorithm. Denote these algorithms by Encod and $\widetilde{\mathsf{Encod}}$, respectively. On a key K and a word/label pair (w, l), the modified encoding algorithm outputs

$$\widetilde{\mathsf{Encod}}_K(w, l) = \Big(\mathsf{Encod}_K(w, l), l\Big) .$$

That is, the label is revealed to the adversary. It is easy to see that $\widetilde{\Pi}$ is still Leak-aIND-secure, because the adversary already knows the labels as he chooses them himself. Therefore, his advantage in attacking $\widetilde{\Pi}$ is the same as that of attacking Π.

However, the modified scheme is not Leak-aSIM-secure, because the leakage profile does not contain any information about the label set. Therefore, the

simulator has no way to simulate the output of $\widetilde{\mathsf{Encod}}$ except to guess. But this will be detected by an adversary who constructs the databases with random labels. We conclude that there exists an attacker that can distinguish the real and ideal worlds for every given simulator. □

The naIND and naSIM security definitions have shown to be equivalent in [20], assuming that the leakage function is label-leaking. Similar to the proof of Proposition 7.5, one can show that this does not hold true for the label-hiding function of our interest in Definition 4.2.

Proposition 7.6 (naIND $\not\Rightarrow$ naSIM) *Let* Leak *be any of the leakage functions defined in Definition 4.2 (i.e,* MinLeak *or* $\mathsf{MonLeak}_k$*). Then, the* Leak-*naIND security does not necessarily imply the* Leak-*naSIM security.*

8 Summary of security implications and separations

Figures 1 and 2 summarizes the results of the previous sections for the minimal and k-monotonous leakage functions respectively.

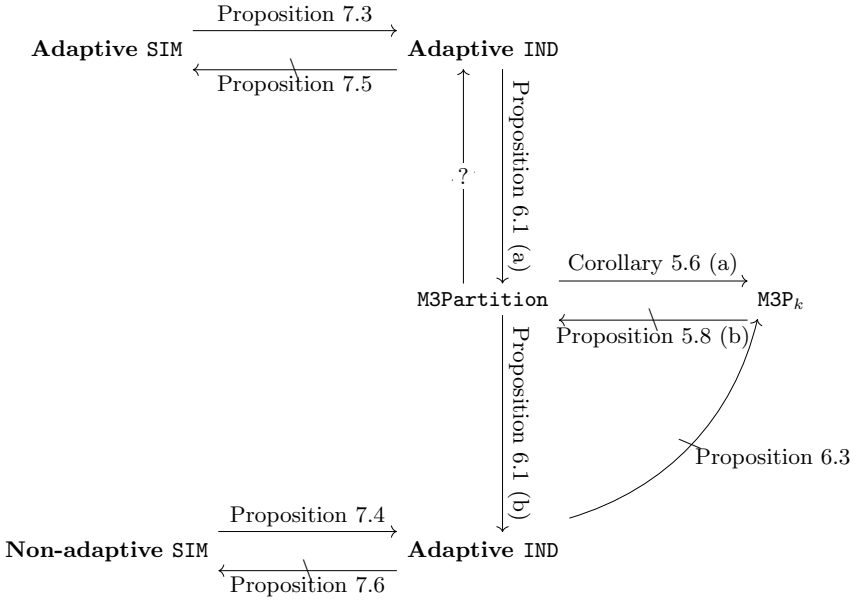

Fig. 1. Relations between different security notions for minimal leakage.

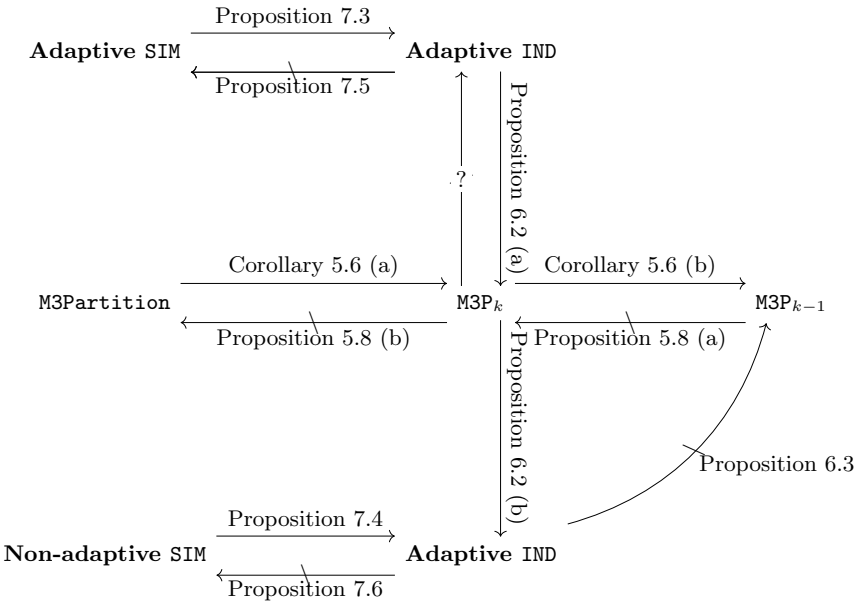

Fig. 2. Relations between different security notions for k-monotonous leakage.

9 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 $\texttt{M3P}_{k+1}$-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 $F : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \mathbb{Z}_q^*$.

9.1 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 9.1), which is a variant of the XDH assumption [3,8,16,25]. The algorithms of our main scheme $\Pi = (\mathsf{Gen}, \mathsf{Encod}, \mathsf{Token}, \mathsf{Adjust})$ are defined as follows:

– **Key generation:** On input 1^λ, run $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, q, e) \leftarrow \mathcal{G}(1^\lambda)$ and choose a label-key $lk \in \{0,1\}^\lambda$ and a word-key $wk \in \{0,1\}^\lambda$ uniformly at random, and output $Param = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, q, e)$ and $K = (lk, wk)$.

– **Encoding algorithm:** On inputs $Param = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, q, e)$, a key $K = (lk, wk)$, a word $w \in \{0,1\}^\lambda$ and a column label $l \in \{0,1\}^\lambda$, output the encoded-word as

$$\widetilde{w} := g_1^{F_{lk}(l) \cdot F_{wk}(w)}.$$

– **Token generation:** On inputs $Param = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, q, e)$, a key $K = (lk, wk)$, and a list of column labels $(l_1, \cdots, l_m)$ with $m \geq 2$, choose random values $r_1, \cdots, r_m \in \mathbb{Z}_q$ subject to $r_1 + \cdots + r_m = 0$ and output 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 $P = \prod_{j=1}^{m} F_{lk}(l_j)$.

– **Adjustment:** On inputs $Param = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, q, e)$, an encoded-word $\widetilde{w}$, and an adjustment token at, output the adjusted word as

$$aw = e(\widetilde{w}, at) \in \mathbb{G}_T.$$

– **Evaluation:** On inputs $Param = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, q, e)$ and a list of adjusted words $(aw_1, \cdots, aw_m)$, output 1 if and only if $\prod_{i=1}^{m} aw_i = e(g_1, g_2)$.

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

$$\begin{aligned} aw_j &= \mathsf{Adjust}_{Param}\big(\mathsf{Encod}_K(w_j, l_j), at_j\big) \\ &= e(g_1^{F_{lk}(l_j) \cdot F_{wk}(w_j)}, g_2^{\frac{P}{F_{lk}(l_j)} \cdot r_j}) = e(g_1, g_2)^{F_{wk}(w_j) \cdot P \cdot r_j}, \end{aligned} \tag{9.1}$$

for every $j \in [m]$, where $(Param, K)$ 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.

9.2 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 9.1). 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:

- **Token generation:** On inputs $Param = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, q, e)$, a key $K = (lk, wk)$, and a list of column labels $(l_1, \cdots, l_m)$ with $m \geq 2$, do as follows. 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.
- **Evaluation:** On inputs $Param = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, q, e)$ and a list of adjusted words $(aw_1, \cdots, aw_m)$ do as follows. 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 (9.1), 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 $j \in \left\{0, k, 2k, 3k, \ldots, \lceil \frac{m-1}{k} \rceil k\right\}$ 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.

9.3 Security analysis

The `M3Partition`-security of our non-monotonous scheme and the $\texttt{M3P}_k$-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 [3,8,16,25].

Assumption 9.1 (Mixed External Diffie-Hellman Variant (MXDHV)) *We say that the MXDHV problem is hard relative to the bilinear map generator $\mathcal{G}$, if for all PPT adversaries $\mathcal{A}$ there exists a negligible function ε such that*

$$
\begin{aligned}
|\Pr[Param \leftarrow \mathcal{G}(1^\lambda);\ a, c, m_0, r', r'' \leftarrow \mathbb{Z}_q^* :\\
\mathcal{A}(Param, g_1^a, g_1^c, g_1^{am_0}, g_1^{cm_0}, g_2^{r'}, g_2^{r''}, g_2^{ar'}, g_2^{cr''}, g_2^{acr'}, g_2^{acr''})] -\\
\Pr[Param \leftarrow \mathcal{G}(1^\lambda);\ a, c, m_0, r, r', r'' \leftarrow \mathbb{Z}_q^* :\\
\mathcal{A}(Param, g_1^a, g_1^c, g_1^{am_0}, g_1^r, g_2^{r'}, g_2^{r''}, g_2^{ar'}, g_2^{cr''}, g_2^{acr'}, g_2^{acr''})]| \leq \varepsilon(\lambda).
\end{aligned}
$$

Whenever a new cryptographic hardness assumption is introduced, it is natural to question its validity. Therefore, it is important to seek for some evidence on the difficulty of the underlying problem. The generic group model, introduced by Shoup [27], provides some confidence on the hardness of group-based assumptions. 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). Therefore, the model assumes that the adversary has not prior information about the internal structure of the group. Gilles et al. [6] have shown that the problem of analyzing assumptions in the generic group model can be reduced to solving problems in polynomial algebra. Using linear algebra tools, they develop 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 available at [1].

Theorem 9.2. *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 9.1 is `M3Partition`-secure.*
- *The proposed k-monotonous construction of Section 9.2 is $\texttt{M3P}_k$-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 9.1 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 9.2 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

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

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 [20], 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 $(Param, g_1^a, g_1^c, g_1^{am_0}, g_1^{am_1}, g_1^{s}, g_2^{r'}, g_2^{r''}, g_2^{ar'}, g_2^{cr''}, g_2^{acr'}, g_2^{acr''})$ as an input, where $s=cm_0$ or $s=cm_1$, 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):

$$f(l)=\begin{cases} a\cdot f_l & l\in\mathscr{L}\\ f_l & l\in\mathscr{M}\\ c\cdot f_l & l\in\mathscr{R}\end{cases}, \tag{9.2}$$

and

$$h(w)=\begin{cases} m_0 & w=w_0^*\\ m_1 & w=w_1^*\\ h_w & w\neq w_0^*,w_1^*\end{cases}, \tag{9.3}$$

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 of the form $(Param, g_1^a, g_1^c, g_1^{am_0}, g_1^{am_1}, g_1^{cm_0}, g_2^{r'}, g_2^{r''}, g_2^{ar'}, g_2^{cr''}, g_2^{acr'}, g_2^{acr''})$ 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 of the form $(Param, g_1^a, g_1^c, g_1^{am_0}, g_1^{am_1}, g_1^{cm_1}, g_2^{r'}, g_2^{r''}, g_2^{ar'}, g_2^{cr''}, g_2^{acr'}, g_2^{acr''})$, 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 [20]. 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

$$|\Pr[\mathsf{Exp}_{\$\mathsf{F},\mathcal{A}}(\lambda, 0) = 1] - \Pr[\mathsf{Exp}_{\$\mathsf{F},\mathcal{A}}(\lambda, 1) = 1]| \tag{9.4}$$

$$= (p(\lambda) + 1)^2 \cdot |\Pr[\mathsf{Chal}^{\mathcal{A}}(X_\lambda) = 1] - \Pr[\mathsf{Chal}^{\mathcal{A}}(Y_\lambda) = 1]| \,. \tag{9.5}$$

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 (9.5) is negligible, and therefore also the expression in Equation (9.4) is negligible as well, completing the proof of the theorem. □

10 Performance analysis

There are two naive solutions for private multiple-join queries, *trivial-client* and *trivial-server*, which are described as follows.

Trivial-client. In this setting, the client does not outsource his database to the server, and stores and computes every thing locally. That is, the algorithms in the plain scenario are used to get the result.

Trivial-server. In this setting, the client encrypts each column separately using a usual CPA-secure symmetric-key encryption scheme and outsources the encrypted database to the server. When he later privately issues join queries for some subset of columns, he fetches back the corresponding encrypted columns, decrypts them and employs the algorithms in the plain scenario to locally compute the result.

Table 2 compares the time and space complexities of our scheme with the naive solutions in terms of the following parameters:

- t, the total number of the columns in the database,
- n, the maximum length of each column in the database,
- m, the number of columns in a join query,
- s, the size of the result set.

In addition, Table 2 also compares leakage profile of the different schemes in terms of the defined leakages in Definition 4.2.

Table 2. Time and space complexity of different approaches for the private set operations

Construction	Storage		Communication		Computation		Leakage				
	Client	Server	Client	Server	Client	Server	SP	DIM	DP	RP	MP_k
Trivial-client	$\mathcal{O}(tn)$	0	0	0	$\mathcal{O}(m \cdot n)$	0	×	×	×	×	×
Trivial-server	$\mathcal{O}(1)$	$\mathcal{O}(tn)$	$\mathcal{O}(m)$	$\mathcal{O}(m \cdot n)$	$\mathcal{O}(m \cdot n)$	$\mathcal{O}(m \cdot n)$	✓	✓	×	×	×
non-monotonous scheme	$\mathcal{O}(1)$	$\mathcal{O}(tn)$	$\mathcal{O}(m)$	$\mathcal{O}(s)$	$\mathcal{O}(s)$	$\mathcal{O}(n^{m-1})$	✓	✓	✓	✓	×
k-monotonous scheme	$\mathcal{O}(1)$	$\mathcal{O}(tn)$	$\mathcal{O}(m)$	$\mathcal{O}(s)$	$\mathcal{O}(s)$	$\mathcal{O}\big((m-1) \cdot n^k/k\big)$	✓	✓	✓	✓	✓

In the following, we provide detailed efficiency comparisons between the two M-Adjoin schemes of [20] and our proposed constructions: the non-monotonous scheme of Section 9.1 and the k-monotonous scheme of Section 9.2.

Table 3 compares all schemes in terms of the underlying hardness assumption, the computational complexity (join time of m columns of length n), storage complexity (encoded word size and token size both in terms of group elements), and the achieved security level.

Table 3. Asymptotic comparison.

Construction	Ref.	Assumption	Join time	Encoded word size (group elements)	Token size (group elements)	Security
M-Adjoin construction I	[20]	Decision Linear [8]	$(m-1)n$	4	$4m$	$\texttt{M3P}_1$ ($\texttt{3Partition}$)
M-Adjoin construction II	[20]	matix-DDH [14]	$(m-1)n$	2	$2m$	$\texttt{M3P}_1$ ($\texttt{3Partition}$)
Our k-monotonous scheme	[Section 9.2]	MXDHV [Assumption 9.1]	$(m-1)n^k/k$	1	m	$\texttt{M3P}_k$
Our non-monotonous scheme	[Section 9.1]	MXDHV [Assumption 9.1]	n^{m-1}	1	m	$\texttt{M3P}_\infty$ ($\texttt{M3Partition}$)

When it comes to concrete performance comparison, our schemes are substantially more efficient both in computation and storage than the previous ones.

The reason for the decrease of overheads is the difference in the employed hardness assumptions. The assumptions in [20] 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 size than the asymmetric ones [5,13]. To be fair, we remark that the M-Adjoin constructions of [20] can be also based on asymmetric bilinear maps, but the assumptions become non-standard.

We have implemented the M-Adjoin scheme II of [20] (both with symmetric and asymmetric pairing) and our non-monotonous M-Adjoin scheme in Java on an Ubuntu 17.04 desktop PC with an Intel Processor 2.9 GHz. We have used the JPBC library [12] for our implementation. Table 4 concretely compares constructions for 128-bit security level. We remark that type-A curves are commonly used for the symmetric pairing settings and type-F curves (also known as BN-curves) are commonly used for constructions under the asymmetric setting. We refer the reader to Appendix B for comparison of group sizes for different security levels.

Table 4. Concrete comparison on a typical processor

Construction	Ref.	Pairing type	Gen (ms)	Encod (ms)	Token (ms)	Adjust (ms)	Encoded-word size (byte)	Token size (byte)
M-Adjoin construction II (symmetric)	[20]	A-curve	32612	1058	1798	1180	768	768
M-Adjoin construction II (asymmetric)	[20]	F-curve	21025	85	394	1089	128	256
Our non-monotonous scheme	[Section 9.1]	F-curve	19052	35	179	1028	64	128

11 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 $\texttt{M3Partition}$ and $\texttt{M3P}_k$ security notions and studied their hierarchical relations. Additionally, we proposed a main scheme that achieves $\texttt{M3Partition}$ 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 providing an efficient M-Adjoin construction that satisfies security against adaptive adversaries, extending the M-Adjoin schemes to the multi-user models, supporting dynamic storage mechanism, extending the M-Adjoin schemes to the case in which the server is malicious.

References

1. https://github.com/generic-group-analyzer/gga (Accessed August 2019)
2. Ateniese, G., Camenisch, J., De Medeiros, B.: Untraceable rfid tags via insubvertible encryption. In: Proceedings of the 12th ACM conference on Computer and communications security. pp. 92–101. ACM (2005)
3. Ballard, L., Green, M., De Medeiros, B., Monrose, F.: Correlation-resistant storage via keyword-searchable encryption. IACR Cryptology ePrint Archive **2005**, 417 (2005)
4. Barbulescu, R., Duquesne, S.: Updating key size estimations for pairings. Journal of Cryptology pp. 1–39 (2017)
5. Barker, E., Barker, W., Burr, W., Polk, W., Smid, M.: Recommendation for key management part 1: General (revision 3). NIST special publication **800**(57), 1–147 (2012)
6. Barthe, G., Fagerholm, E., Fiore, D., Mitchell, J., Scedrov, A., Schmidt, B.: Automated analysis of cryptographic assumptions in generic group models. In: Annual Cryptology Conference. pp. 95–112. Springer (2014)
7. Blakley, G.R.: Safeguarding cryptographic keys. Proc. of the National Computer Conference1979 **48**, 313–317 (1979)
8. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: Annual International Cryptology Conference. pp. 41–55. Springer (2004)
9. Camenisch, J., Hohenberger, S., Lysyanskaya, A.: Compact e-cash. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 302–321. Springer (2005)
10. Chatterjee, S., Menezes, A.: On cryptographic protocols employing asymmetric pairingsthe role of ψ revisited. Discrete Applied Mathematics **159**(13), 1311–1322 (2011)
11. Curtmola, R., Garay, J.A., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006. pp. 79–88 (2006). https://doi.org/10.1145/1180405.1180417, https://doi.org/10.1145/1180405.1180417
12. De Caro, A., Iovino, V.: jpbc: Java pairing based cryptography. In: Proceedings of the 16th IEEE Symposium on Computers and Communications, ISCC 2011. pp. 850–855. Kerkyra, Corfu, Greece, June 28 - July 1 (2011)
13. ECRYPT, I.: Yearly report on algorithms and key lengths (2010) (2011)
14. Escala, A., Herold, G., Kiltz, E., Rafols, C., Villar, J.: An algebraic framework for diffie–hellman assumptions. Journal of cryptology **30**(1), 242–288 (2017)
15. Furukawa, J., Isshiki, T.: Controlled joining on encrypted relational database. In: International Conference on Pairing-Based Cryptography. pp. 46–64. Springer (2012)
16. Galbraith, S.D., Rotger, V.: Easy decision diffie-hellman groups. LMS Journal of Computation and Mathematics **7**, 201–218 (2004)
17. Hang, I., Kerschbaum, F., Damiani, E.: Enki: access control for encrypted query processing. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. pp. 183–196. ACM (2015)
18. Ito, M., Saito, A., Nishizeki, T.: Secret sharing scheme realizing general access structure. Electronics and Communications in Japan (Part III: Fundamental Electronic Science) **72**(9), 56–64 (1989)

19. Kamara, S., Moataz, T.: Sql on structurally-encrypted databases. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 149–180. Springer (2018)
20. Mironov, I., Segev, G., Shahaf, I.: Strengthening the security of encrypted databases: non-transitive joins. In: Theory of Cryptography Conference. pp. 631–661. Springer (2017)
21. Popa, R.A.: Building practical systems that compute on encrypted data. Ph.D. thesis, Massachusetts Institute of Technology, Department of Electrical Engineering (2014)
22. Popa, R.A., Redfield, C., Zeldovich, N., Balakrishnan, H.: Cryptdb: protecting confidentiality with encrypted query processing. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. pp. 85–100. ACM (2011)
23. Popa, R.A., Redfield, C., Zeldovich, N., Balakrishnan, H.: Cryptdb: processing queries on an encrypted database. Communications of the ACM **55**(9), 103–111 (2012)
24. Popa, R.A., Zeldovich, N.: Cryptographic treatment of cryptdb's adjustable join (2012)
25. Scott, M.: Authenticated id-based key exchange and remote log-in with simple token and pin number. IACR Cryptology ePrint Archive **2002**, 164 (2002)
26. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979), `http://doi.acm.org/10.1145/359168.359176`
27. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 256–266. Springer (1997)

A The challenger of the proof of Theorem 9.2

In this section, we describe the challenger mentioned in the proof of Theorem 9.2 in details. We first describe the challenger for proving claim I (i.e., `M3Partition`-security of our non-autonomous schemes). At the end we will discuss the case of claim II.

1. **Setup phase:** The challenger provides adversary $\mathcal{A}$ with public parameters $Param$.
2. **Pre-challenge query phase:** In the following, we determine how the challenger handles the adversary's queries:
   (a) $\mathsf{Addlbl}(l, X)$**:** The challenger adds the column label l to the group X, where $X \in \{\mathscr{L}, \mathscr{M}, \mathscr{R}\}$. In addition, the challenger chooses $f_l \leftarrow \mathbb{Z}_q^*$, unless it has already been sampled. The value f_l is used by the challenger only for his own future use in the computation of the adjustment encoded-words and the adjustment tokens. As it was mentioned above, the idea is to simulate $f(l)$ in $\mathsf{Exp}_{\$\mathsf{F},\mathcal{A}}$ as in Equation (9.2).
   (b) $\mathsf{Encod}_K(w, l)$**:** For the word w, the challenger chooses $h_w \leftarrow \mathbb{Z}_q^*$, unless it has already been sampled. Then the challenger computes $\widetilde{w}$ as follows

$$\widetilde{w} = \begin{cases} (g_1^a)^{f_l \cdot h_w} & l \in \mathscr{L} \\ (g_1)^{f_l \cdot h_w} & l \in \mathscr{M} \\ (g_1^c)^{f_l \cdot h_w} & l \in \mathscr{R} \end{cases}.$$

It then returns $\widetilde{w}$ to the adversary $\mathcal{A}$. Indeed, the challenger computes $\widetilde{w}$ such that $f(l)$ and $h(w)$ in $\mathsf{Exp}_{\$\mathsf{F},\mathcal{A}}$ correspond to Equations (9.2) and (9.3).

(c) $\mathsf{Token}_K(l_1,\cdots,l_m)$**:** The challenger chooses random values $r_1,\cdots,r_m \in \mathbb{Z}_q$ such that $r_1+\cdots+r_m=0$. For each $j\in[m]$, the challenger computes at_j as

$$at_j = (X_j)^{\frac{\prod_{i=1}^{m} f_{l_i}}{f_{l_j}}\cdot r_j}, \tag{A.1}$$

where $X_j \in \{g_2^{r'}, g_2^{r''}, g_2^{ar'}, g_2^{cr''}, g_2^{acr'}, g_2^{acr''}\}$. In the following, we explain how $X_1,\cdots,X_m$ are chosen in each case:

- **Case 1:** $l_1,\cdots,l_m$ all belong to $\mathscr{L}$, $\mathscr{M}$, or $\mathscr{R}$: In this case, we let $X_j = g_2^{r''}$ for every $j\in[m]$. To see why this works, consider the case where $l_1,\cdots,l_m\in\mathscr{L}$. The other cases are similar. We have

$$at_j = (X_j)^{\frac{\prod_{i=1}^{m} f_{l_i}}{f_{l_j}} r_j} = (g_2^{r''})^{\frac{\prod_{i=1}^{m} af_{l_i}}{af_{l_j}} a^{-(m-1)} r_j} = g_2^{\frac{\prod_{i=1}^{m} f(l_i)}{f(l_j)} z_j}.$$

  As it can be seen, the actual randomness used in computation of at_j is $z_j = a^{-(m-1)}r''r_j$.
- **Case 2:** $l_1,\cdots,l_m \in \mathscr{L}\cup\mathscr{M}$ and the intersection of $\{l_1,\cdots,l_m\}$ with both $\mathscr{L}$ and $\mathscr{M}$ is non-empty. In this case, we let

$$X_j = \begin{cases} g_2^{r'} & l_j\in\mathscr{L} \\ g_2^{ar'} & l_j\in\mathscr{M} \end{cases}.$$

  We have

$$\begin{aligned} at_j &= (X_j)^{\frac{\prod_{i=1}^{m} f_{l_i}}{f_{l_j}} r_j} \\ &= \begin{cases} (g_2^{r'})^{\frac{(\prod_{l_i\in\mathscr{L}} af_{l_i})(\prod_{l_i\in\mathscr{M}} f_{l_i})}{af_{l_j}} a^{-(n_1-1)} r_j} & l_j\in\mathscr{L} \\ (g_2^{ar'})^{\frac{(\prod_{l_i\in\mathscr{L}} af_{l_i})(\prod_{l_i\in\mathscr{M}} f_{l_i})}{f_{l_j}} a^{-n_1} r_j} & l_j\in\mathscr{M} \end{cases} \\ &= g_2^{\frac{\prod_{i=1}^{m} f(l_i)}{f(l_j)} z_j} \end{aligned}$$

  where n_1 is the number of the labels in the list $(l_1.\cdots,l_m)$ that appear in $\mathscr{L}$, and $z_j = a^{-(n_1-1)}r'r_j$ for every $j\in[m]$.

– **Case 3:** $l_1, \cdots, l_m \in \mathscr{M} \cup \mathscr{R}$ and the intersection of $\{l_1, \cdots, l_m\}$ with both $\mathscr{M}$ and $\mathscr{R}$ is non-empty. In this case, we let

$$X_j = \begin{cases} g_2^{cr''} & l_j \in \mathscr{M} \\ g_2^{r''} & l_j \in \mathscr{R} \end{cases}.$$

This choice can be justified similar to case 2.

– **Case 4:** $l_1, \cdots, l_m \in \mathscr{L} \cup \mathscr{M} \cup \mathscr{R}$ and the intersection of $\{l_1, \cdots, l_m\}$ with each of $\mathscr{L}, \mathscr{M}$ and $\mathscr{R}$ is non-empty. Since $\{l_1, \cdots, l_m\} \cap \mathscr{M} \neq \emptyset$, let p be the smallest integer such that $l_p \in \{l_1, \cdots, l_m\} \cap \mathscr{M}$. We compute at_j for $j \in [m] \backslash \{p\}$ as in Equation (A.1) but at_p is computed differently. For $j \in [m] \backslash \{p\}$, we let

$$X_j = \begin{cases} g_2^{cr''} & l_j \in \mathscr{L} \\ g_2^{acr'} & l_j \in \mathscr{M},\ l_j \neq l_p \\ g_2^{ar'} & l_j \in \mathscr{R} \end{cases}.$$

We compute at_p as follows

$$\begin{aligned} at_p = & \prod_{l_k \in \mathscr{L}} (g_2^{-acr''})^{\frac{\prod_{i=1}^{m} f_{l_i}}{f_{l_p}} \cdot r_k} \times \prod_{l_k \neq l_p \in \mathscr{M}} (g_2^{-acr'})^{\frac{\prod_{i=1}^{m} f_{l_i}}{f_{l_p}} \cdot r_k} \\ & \times \prod_{l_k \in \mathscr{R}} (g_2^{-acr'})^{\frac{\prod_{i=1}^{m} f_{l_i}}{f_{l_p}} \cdot r_k}. \end{aligned} \tag{A.2}$$

The reason for the choices X_j, $j \in [m] \backslash \{p\}$, and at_p is justified next. Notice that for $j \in [m] \backslash \{p\}$, we have

$$at_j = (X_j)^{\frac{\prod_{i=1}^{m} f_{l_i}}{f_{l_j}} r_j}$$

$$= \begin{cases} (g_2^{cr''})^{\frac{(\prod_{l_i \in \mathscr{L}} a f_{l_i})(\prod_{l_i \in \mathscr{M}} f_{l_i})(\prod_{l_i \in \mathscr{R}} c f_{l_i})}{a f_{l_j}} a^{-(n_1-1)} c^{-n_3} r_j} & l_j \in \mathscr{L} \\ (g_2^{acr'})^{\frac{(\prod_{l_i \in \mathscr{L}} a f_{l_i})(\prod_{l_i \in \mathscr{M}} f_{l_i})(\prod_{l_i \in \mathscr{R}} c f_{l_i})}{f_{l_j}} a^{-n_1} c^{-n_3} r_j} & l_j \in \mathscr{M},\ l_j \neq l_p \\ (g_2^{ar'})^{\frac{(\prod_{l_i \in \mathscr{L}} a f_{l_i})(\prod_{l_i \in \mathscr{M}} f_{l_i})(\prod_{l_i \in \mathscr{R}} c f_{l_i})}{c f_{l_j}} a^{-n_1} c^{-(n_3-1)} r_j} & l_j \in \mathscr{R} \end{cases},$$

$$= g_2^{\frac{\prod_{i=1}^{m} f(l_i)}{f(l_j)} z_j},$$

where n_1, n_3 are the number of the labels in the list $(l_1.\cdots, l_m)$ that appear in $\mathscr{L}$ and $\mathscr{R}$, respectively. Therefore, the actual randomness used in computing at_j is as follows

$$z_j = \begin{cases} (a^{(n_1-1)}c^{(n_3-1)}r''r_j) & l_j \in \mathscr{L} \\ (a^{(n_1-1)}c^{(n_3-1)}r'r_j) & l_j \in \mathscr{M} \cup \mathscr{R},\ l_j \neq l_p \end{cases}. \tag{A.3}$$

It remains to show that

$$at_p = g_2^{\frac{(\prod_{l_i\in\mathscr{L}} af_{l_i})(\prod_{l_i\in\mathscr{M}} f_{l_i})(\prod_{l_i\in\mathscr{R}} cf_{l_i})}{f_{l_p}} z_p},$$

for some z_p where $z_p = -\sum_{j\in[m]\setminus\{p\}} z_j$. We can rewrite Equation (A.2) as:

$$\begin{aligned}
at_p &= \prod_{l_k\in\mathscr{L}} (g_2^{-acr''})^{\frac{(\prod_{l_i\in\mathscr{L}} af_{l_i})(\prod_{l_p\neq l_i\in\mathscr{M}} f_{l_i})(\prod_{l_i\in\mathscr{R}} cf_{l_i})}{f_{l_p}} a^{-(n_1-1)}c^{-(n_3-1)}r_j} \\
&\times \prod_{l_k\neq l_p\in\mathscr{M}} (g_2^{-acr'})^{\frac{(\prod_{l_i\in\mathscr{L}} af_{l_i})(\prod_{l_p\neq l_i\in\mathscr{M}} f_{l_i})(\prod_{l_i\in\mathscr{R}} cf_{l_i})}{f_{l_p}} a^{-(n_1-1)}c^{-(n_3-1)}r_j} \\
&\times \prod_{l_k\in\mathscr{R}} (g_2^{-acr'})^{\frac{(\prod_{l_i\in\mathscr{L}} af_{l_i})(\prod_{l_p\neq l_i\in\mathscr{M}} f_{l_i})(\prod_{l_i\in\mathscr{R}} cf_{l_i})}{f_{l_p}} a^{-(n_1-1)}c^{-(n_3-1)}r_j} \\
&= \prod_{l_k\in\mathscr{L}} g_2^{-\frac{\prod_{i=1}^m f(l_i)}{f(l_p)} z_k} \times \prod_{l_p\neq l_k\in\mathscr{M}} g_2^{-\frac{\prod_{i=1}^m f(l_i)}{f(l_p)} z_k} \times \prod_{l_k\in\mathscr{R}} g_2^{-\frac{\prod_{i=1}^m f(l_i)}{f(l_p)} z_k} \\
&= g_2^{-\frac{\prod_{i=1}^m f(l_i)}{f(l_p)}(-\sum_{j\in[m]\setminus\{p\}} z_j)}.
\end{aligned}$$

The claim then follows.

3. **Challenge phase:** The adversary $\mathcal{A}$ chooses a pair of challenge words w_0^*, w_1^*. Then, the challenger returns an encoded-word $\widetilde{w} = (g_1^s)^{f_l}$, for every $l \in \mathscr{R}$, to the adversary $\mathcal{A}$.
4. **Post-challenge query phase:** The challenger Chal responds as in the pre-challenge phase, but by definition of the `M3Partition` security, it must also handle:
   - $\mathsf{Encod}(w, l)$: In case that $l \in \mathscr{L}$ and $w = w_0^*$ or $w = w_1^*$, the challenger returns the encoded-word $(g_1^{am_0})^{f_l}$ or $(g_1^{am_1})^{f_l}$ to the adversary $\mathcal{A}$, respectively.
   - $\mathsf{AddIbl}(l, X)$: In case that $X = \mathscr{R}$, the challenger provides the adversary $\mathcal{A}$ with $(g_1^s)^{f_l}$.

5. **Output phase:** The challenger outputs the value $\sigma \in \{0,1\}$ that adversary $\mathcal{A}$ outputs.

Description of the challenger for proving claim II. The challenger is exactly the same as before, except that the way a $\mathsf{Token}_K(l_1, \cdots, l_m)$ query is answered is modified as follows.

Notice that by definition of the $\mathtt{M3P}_k$-security, we only need to answer queries when $2 \leq m \leq k+1$. If $2 \leq m \leq k$, we do as before, assuming the simple (m,m)-threshold scheme of the non-monotonous case is used. If not, some minor modifications are required, which we ignore to discuss since it is similar to the case where $m = k+1$ and will be described next. The challenger chooses $r_1, \cdots, r_m \in \mathbb{Z}_q$ by sharing $0 \in \mathbb{Z}_q$ using the $(m, m-1)$-threshold scheme and computes at_j as before, i.e., using Equation (A.1). Recall that we had cases for explaining how to choose $X_1, \cdots, X_m$. Cases 1-3 are as before. The reason that it works is that $z_1, \ldots, z_m$ are also random shares of the value 0 using the $(m, m-1)$-threshold scheme. Case 4 is the more complicated one. Again, let p be the smallest integer such that $l_p \in \{l_1, \cdots, l_m\} \cap \mathscr{M}$. For each $j \in [m] \backslash \{p\}$, choose r_j at compute at_j as before. That is:

$$at_j = (X_j)^{\frac{\prod_{i=1}^{m} f_{l_i}}{f_{l_j}} r_j} = g_2^{\frac{\prod_{i=1}^{m} f(l_i)}{f(l_j)} z_j}$$

Where z_j, $j \in [m] \backslash \{p\}$, is as in Equation (A.3). We need to compute at_p such that

$$at_p = g_2^{\frac{\prod_{i=1}^{m} f(l_i)}{f(l_j)} z_p}$$

and $z_1, \ldots, z_m$ are random shares of the value 0 using the $(m, m-1)$-threshold scheme. It is easy to verify that it is sufficient to compute at_p as follows:

$$\begin{aligned} at_p = & \prod_{l_k \in \mathscr{L}} (g_2^{\gamma_k acr''})^{\frac{\prod_{i=1}^{m} f_{l_i}}{f_{l_p}} \cdot r_k} \times \prod_{l_k \neq l_p \in \mathscr{M}} (g_2^{\gamma_k acr'})^{\frac{\prod_{i=1}^{m} f_{l_i}}{f_{l_p}} \cdot r_k} \\ & \times \prod_{l_k \in \mathscr{R}} (g_2^{\gamma_k acr'})^{\frac{\prod_{i=1}^{m} f_{l_i}}{f_{l_p}} \cdot r_k}. \end{aligned} \tag{A.4}$$

where $\gamma_k = -\alpha_p^{-1} \alpha_k$, where $\alpha_1, \ldots, \alpha_m$ are the constants for reconstructing the secret from the shares.

The above explanation completes the description of the challenger for proving both claim I and II. 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 of

the form $(Param, g_1^a, g_1^c, g_1^{am_0}, g_1^{am_1}, g_1^{cm_0}, g_2^{r'}, g_2^{r''}, g_2^{ar'}, g_2^{cr''}, g_2^{acr'}, g_2^{acr''})$ as 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 a sample of Y_λ of the form $(Param, g_1^a, g_1^c, g_1^{am_0}, g_1^{am_1}, g_1^{cm_1}, g_2^{r'}, g_2^{r''}, g_2^{ar'}, g_2^{cr''}, g_2^{acr'}, g_2^{acr''})$ as input, 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 [20]. 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

$$|\Pr[\mathsf{Exp}_{\$\mathsf{F},\mathcal{A}}(\lambda, 0) = 1] - \Pr[\mathsf{Exp}_{\$\mathsf{F},\mathcal{A}}(\lambda, 1) = 1]| \tag{A.5}$$

$$= (p(\lambda) + 1)^2 \cdot |\Pr[\mathsf{Chal}^{\mathcal{A}}(X_\lambda) = 1] - \Pr[\mathsf{Chal}^{\mathcal{A}}(Y_\lambda) = 1]| \,. \tag{A.6}$$

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 (A.6) is negligible, and therefore also the expression in Equation (A.5) is negligible as well, completing the proof of the theorem. □

B Pairing types and security levels

An appropriate pairing for cryptographic applications requires that the discrete logarithm problem be sufficiently difficult on the groups defined by the bilinear map generator $\mathcal{G}$. Three kinds of pairings are generally recognized in the cryptographic literature:

- **Type-1:** $\mathbb{G}_1 = \mathbb{G}_2$,
- **Type-2:** $\mathbb{G}_1 \neq \mathbb{G}_2$ and there is an efficiently computable isomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$,
- **Type-3:** $\mathbb{G}_1 \neq \mathbb{G}_2$ and there are no efficiently computable isomorphisms between $\mathbb{G}_2$ and $\mathbb{G}_1$.

The Type-3 pairing, among the different types of pairings, provides the most compact parameter sizes and the most efficient algorithms [10]. In contrast, since the Type-1 pairings are usually defined over low characteristics fields, there is a series of attacks on them, making them unusable [4].

We say that a pairing is symmetric if $\mathbb{G}_1 = \mathbb{G}_2$, else it is asymmetric. Type-A curves are commonly used for the symmetric pairing settings. Type-F curves (also known as BN-curves) are commonly used for constructions under the asymmetric setting. In Table 5, we present key and group sizes for pairing-based schemes based on different security levels, in accordance to the standards provided in [5,13].

Table 5. Key and group sizes

Security level	Protection period	AES key	Type A-curve			Type F-curve			
			$\mathbb{G}_1 = \mathbb{G}_2$	$\mathbb{G}_T$	$\mathbb{Z}_q$	$\mathbb{G}_1$	$\mathbb{G}_2$	$\mathbb{G}_T$	$\mathbb{Z}_q$
80-bit	Legacy	128	1024	1024	160	320	640	1920	160
112-bit	2011–2030	128	2048	2048	224	448	896	2688	224
128-bit	2030–2040	128	3072	3072	256	512	1024	3072	256
192-bit	>2030	192	7696	7096	384	1280	2560	7680	640
256-bit	>2030	256	15360	15360	512	2560	5120	15360	1280