How (Not) to Index Order Revealing Encrypted Databases

Luca Ferretti[1,*,†], Mattia Trabucco[1,†], Mauro Andreolini[1] and Mirco Marchetti[2]

[1] *University of Modena and Reggio Emilia, Department of Physics, Informatics, Mathematics*
[2] *University of Modena and Reggio Emilia, Department of Engineering "Enzo Ferrari"*

Abstract

Order Reveling Encryption (ORE) enables efficient range queries on encrypted databases, but may leak information that could be exploited by inference attacks. State-of-the-art ORE schemes claim different security guarantees depending on the adversary attack surface. Intuitively, *online adversaries* who access the database server at runtime may access information leakage; *offline adversaries* who access only a snapshot of the database data should not be able to gain useful information. We focus on *offline security* of the ORE scheme proposed by Lewi and Wu (LW-ORE, CCS 2016), which guarantees semantic security of ciphertexts stored in the database, but requires that ciphertexts are maintained sorted with regard to the corresponding plaintexts to support sublinear time queries. The design of LW-ORE does not discuss how to build indexing data structures to maintain sorting. The risk is that practitioners consider indexes as a technicality whose design does not affect security. We show that indexes can affect offline security of LW-ORE because they may leak duplicate plaintext values, and statistical information on plaintexts distribution and on transactions history. As a real-world demonstration, we found two open source implementations related to academic research (JISA 2018, VLDB 2019), and both adopt standard search trees which may introduce such vulnerabilities. We discuss necessary conditions for indexing data structures to be secure for ORE databases, and we outline practical solutions. Our analyses could represent an insightful lesson in the context of security failures due to gaps between theoretical modeling and actual implementation, and may also apply to other cryptographic techniques for securing outsourced databases.

Keywords

Encrypted Database, Order Revealing Encryption, Property Preserving Encryption, Secure Indexes

1. Introduction

Standard encryption solutions for databases protect *data at rest* [1], but require database servers to decrypt data at runtime to execute queries. Two decades ago, research began

ITASEC 2023: The Italian Conference on CyberSecurity, May 03–05, 2023, Bari, Italy
*Corresponding author.
† These authors contributed equally.
EMAIL: luca.ferretti@unimore.it (L. Ferretti); mattia.trabucco@unimore.it (M. Trabucco); mauro.andreolini@unimore.it (M. Andreolini); mirco.marchetti@unimore.it (M. Marchetti)
ORCID: 0000-0001-5824-2297 (L. Ferretti); 0009-0002-5714-6927 (M. Trabucco); 0000-0003-3671-6927 (M. Andreolini); 0000-0002-7408-6906 (M. Marchetti)

investigating how to encrypt *data in use* to execute database queries without decrypting data at the server side, such that only (trusted) clients which possess secret keys only decrypt queries results [2]. First motivated by the advent of database outsourcing and then by public Cloud computing services, encryption of *data in use* would increase data privacy against potentially *curious* service providers [3]. Nowadays, encryption of *data in use* is also relevant for improving security of on-premise solutions in case of data breaches caused by cyber attacks that subvert the database systems or software applications running on top of them [4].

Research efforts for encryption of *data in use* include multifold approaches [5], such as new cryptographic primitives and schemes [6, 7, 8], techniques for aggregating and indexing encrypted data for efficient confidential retrieval [2, 9, 10], key management strategies to support cryptographic access control [11, 12], and data distribution strategies based on secret sharing techniques [13]. Practitioners, including researchers and companies, are proposing implementations which combine a mixture of techniques and additional specialized strategies for supporting specific database environments and improving performance for popular workloads [14, 15, 16]. Parallel research efforts focus on analyzing security, and include novel attack techniques such as inference of statistical information, correlation among multiple data, and analyses of query patterns [17, 18, 19, 20, 21, 22, 23]. Alternative proposals rely on trusted enclave technologies for fully fledged computation, which theoretically achieve better performance and flexibility [24], but in practice showed to suffer from side-channel attacks [25].

We are interested in *Property Preserving Encryption* (PPE) [26], which is a family of cryptographic schemes where ciphertexts support the evaluation of some properties related to their corresponding plaintexts, at the cost of exposing "limited" information leakage, such as the result of the evaluation and potentially additional information that is specific for each scheme. The advantage over *homomorphic encryption* [6], which produces ciphertexts supporting the execution of computation while guaranteeing at least semantic (IND-CPA) security, is enabling practical performance even for quite complex properties. The disadvantage is that information leakage may allow practical attacks on encrypted data, which may also depend on the characteristics of the datasets [19, 20] and on query workloads [18, 23]. Thus, PPE should be carefully adopted or considered as a hardening technique that should not be relied upon as a first defense layer.

We focus on *Order Revealing Encryption* (ORE) [27, 28, 8], which denotes a family of PPE schemes specialized on comparison of numerical data (e.g., "greater than" operators), with consequential support for other derived operations of great interest for many database workloads, such as sorting and substring prefix/suffix match. We analyze the ORE scheme proposed by Lewi and Wu (LW-ORE) [8], which is one of the most important schemes due to good security-performance trade-offs. The peculiar trait of LW-ORE is to include two types of encryption functions denoted as *right encryption* and *left encryption*, each generating a different type of ciphertext from the same plaintext: semantically-secure *right ciphertexts* which are stored in the encrypted database, and deterministic *left ciphertexts* which are sent when querying the database. LW-ORE uses an evaluation function that compares *left* and *right ciphertexts* and output the result (e.g., 0 or 1 if the plaintext used to generate the *left ciphertext* is equal or greater than that used to generate the

right ciphertext). This approach allows to distinguish information leakage against two security models: *offline security*, where adversaries only access a snapshot of the database including *right ciphertexts*, and *online security*, where adversaries also access queries including *left ciphertexts*. While an *offline* attacker should not learn any information (except for the size of the database and of each plaintext) because *right ciphertexts* are semantically (IND-CPA) secure, an *online* attacker could get more information because *left ciphertexts* are deterministic.

In this paper, we reconsider *offline security* of LW-ORE when deployed in real database systems. Although *right ciphertexts* are proved semantically secure, they have to be maintained *sorted* to enable sublinear query times via binary search. However, LW-ORE does not describe indexing data structures or other mechanisms. Thus, the scheme can be straightforwardly applied only to a handful of use cases or should be adopted with only support for linear time queries. In any other scenario, LW-ORE must be integrated with indexing data structures, and the risk is to consider this integration a *technicality*, that is, something that cannot affect the original security guarantees of the scheme. As already experienced many times in applied cryptography and cryptographic engineering, details left open in theoretical designs may open to security vulnerabilities when implemented in real systems. We found relevant literature associated with open source implementations of LW-ORE that indeed adopt standard database indexing techniques based on AVL trees [29] and B+trees [30]. We also found an additional open implementation of the client-side LW-ORE, which however do not describe indexing techniques at the server side [31].

In this paper, we show that using standard indexing techniques for ORE may introduce at least two classes of vulnerabilities:

- the first class of vulnerability is the simpler but also the most severe, and is related to the disclosure of duplicate values: standard indexes store pointers to all duplicate values within the same data structure node, thus *offline security* guarantees of the encrypted database fall back to that of a deterministic encryption scheme, re-enabling powerful *inference attacks*. This vulnerability thus breaks one of the major contributions of LW-ORE, that is, of guaranteeing semantic security of ciphertexts. Ad-hoc variants of standard indexing techniques must be implemented to avoid storing duplicate value within the same data structure node while maintaining the data structure efficient.
- the second class of vulnerability is related to the disclosure of information regarding the history of the transactions executed on the database. This vulnerability may also cause disclosure of information related to the plaintext distribution if such distribution is not independent of the insertion order (or on derived information, such as insertion time). This requirement has been outlined by literature which denote as *history independent* the data structure that do not leak any information about the transactions history that generated the data structure [32, 33]. However, to the best of our knowledge, it is not considered by literature related to database outsourcing and PPE.

The two classes of vulnerabilities may not apply depending on the characteristics of

the database and of the query workload, but protecting against both may require the design of ad-hoc indexing data structures. The last contribution of the paper is to outline such design, leaving further analyses and evaluations as future work.

The rest of the paper is organized as follows. Section 2 outlines ORE schemes and databases. Section 3 discusses possible information leakage due indexing data structures. Section 4 outlines necessary design choices for secure indexes on ORE databases. Section 5 concludes the paper and discusses future work.

2. Analysis of Order Revealing Encryption

The Order Revealing Encryption scheme by Lewi and Wu [8] (LW-ORE) is a symmetric encryption scheme for supporting comparison between encrypted data. First, we describe the operations framework of LW-ORE and outline the range query protocol built on top of it as defined in the original paper [8]. Then, we discuss the limitations of the original models for real database systems and outline better variants.

LW-ORE scheme operations framework. The LW-ORE scheme includes four algorithms: *setup* ($\mathsf{ORE.Setup}$), *left encrypt* ($\mathsf{ORE.Encrypt_L}$), *right encrypt* ($\mathsf{ORE.Encrypt_R}$), and *compare* ($\mathsf{ORE.Compare}$):

- $\mathsf{sk} \xleftarrow{\$} \mathsf{ORE.Setup}(1^\lambda)$ is a probabilistic algorithm which takes security parameter 1^λ and outputs secret key sk;
- $\mathsf{c^L} \leftarrow \mathsf{ORE.Encrypt_L}(\mathsf{sk}, \mathsf{p})$ is a deterministic algorithm which takes key sk and plaintext p and outputs *left ciphertext* $\mathsf{c^L}$;
- $\mathsf{c^R} \xleftarrow{\$} \mathsf{ORE.Encrypt_R}(\mathsf{sk}, \mathsf{p})$ is a probabilistic algorithm which takes key sk and plaintext p and outputs *right ciphertext* $\mathsf{c^R}$;
- $\{-1, 0, 1\} \leftarrow \mathsf{ORE.Compare}(\mathsf{c_1^L}, \mathsf{c_2^R})$ is a deterministic algorithm which takes *left* and *right ciphertexts* $\mathsf{c_1^L}$ and $\mathsf{c_2^R}$, and outputs -1, 0, 1 if p_1 is less then, equal, or greater than p_2, where $\mathsf{c_1^L} \leftarrow \mathsf{ORE.Encrypt_L}(\mathsf{sk}, \mathsf{p}_1)$ and $\mathsf{c_2^R} \xleftarrow{\$} \mathsf{ORE.Encrypt_R}(\mathsf{sk}, \mathsf{p}_2)$, for any $\mathsf{sk} \xleftarrow{\$} \mathsf{ORE.Setup}(1^\lambda)$.

LW-ORE does not expose a native decryption routine, but decryption can be implemented as a binary search over the plaintext domain via encryption and compare (see [8, Remark 2.1]). In the context of encrypted databases, ORE ciphertexts can be associated with ciphertexts computed with any standard symmetric schemes to enable more efficient decryption.

LW-ORE range query protocol for encrypted databases. LW-ORE scheme allows designing a *stateless* and *single round* Range Query protocol (RQ) [8, Section 5.2] among a *client* and a database *server* that includes four operations: database *setup* ($\mathsf{RQ.Setup}$), *range* query ($\mathsf{RQ.Range}$), *insert* ($\mathsf{RQ.Insert}$), and *delete* ($\mathsf{RQ.Delete}$). Without loss of generality, in this paper we omit *delete* to avoid too much verbosity.

- $\mathsf{st} \xleftarrow{\$} \mathsf{RQ.Setup}(1^\lambda, P)$: the client initializes ORE secret key $\mathsf{sk} \xleftarrow{\$} \mathsf{ORE.Setup}(1^\lambda)$, sorts the database P as $\hat{P} = \langle \mathsf{p}_1, \ldots, \mathsf{p}_N \rangle$ such that $\mathsf{p}_n \leq \mathsf{p}_{n+1}\ \forall n \in [N-1]$, and encrypts the sorted database $\hat{P}$ as $\hat{C} = \langle \mathsf{c}^{\mathsf{R}}_1, \ldots, \mathsf{c}^{\mathsf{R}}_N \rangle$ such that $\mathsf{c}^{\mathsf{R}}_n \xleftarrow{\$} \mathsf{ORE.Encrypt}_{\mathsf{R}}(\mathsf{sk}, \mathsf{p}_n)\ \forall n \in [N]$. The client sends $\hat{C}$ to the server, which sets its state information $\mathsf{st} = \hat{C}$.
- $\langle \mathsf{p}_i, \ldots, p_j \rangle \leftarrow \mathsf{RQ.Range}(\mathsf{sk}, \mathsf{q} = (x, y), \mathsf{st} = \hat{C})$: the client computes the tuple $\langle \mathsf{c}^{\mathsf{L}}_x, \mathsf{c}^{\mathsf{L}}_y \rangle$ such that $\mathsf{c}^{\mathsf{L}}_x \leftarrow \mathsf{ORE.Encrypt}_{\mathsf{L}}(\mathsf{sk}, x)$ and $\mathsf{c}^{\mathsf{L}}_y \leftarrow \mathsf{ORE.Encrypt}_{\mathsf{L}}(\mathsf{sk}, y)$, and sends it to the server. The server uses $\mathsf{c}^{\mathsf{L}}_x$ ($\mathsf{c}^{\mathsf{L}}_y$) to find $\mathsf{c}^{\mathsf{R}}_i$ ($\mathsf{c}^{\mathsf{R}}_j$) within $\hat{C}$ such that i (j) is the smallest (greatest) index of $\hat{C} = \langle \mathsf{c}^{\mathsf{R}}_n \rangle_{n \in [N]}$ such that $\mathsf{ORE.Compare}(\mathsf{c}^{\mathsf{L}}_x, \mathsf{c}^{\mathsf{R}}_i) = -1$ ($\mathsf{ORE.Compare}(\mathsf{c}^{\mathsf{L}}_y, \mathsf{c}^{\mathsf{R}}_j) = 1$). As suggested in the original protocol [8], the server can use binary search to compute a number of $\mathsf{ORE.Compare}$ operations that is logarithmic in the size of the database N. The server then returns the slice of values within $\hat{C}$ between $\left\langle \mathsf{c}^{\mathsf{R}}_i, \ldots, \mathsf{c}^{\mathsf{R}}_j \right\rangle$, which the client can decrypt to $\langle \mathsf{p}_i, \ldots, \mathsf{p}_j \rangle$.
- $\mathsf{st}' \xleftarrow{\$} \mathsf{RQ.Insert}(\mathsf{sk}, \mathsf{q} = x, \mathsf{st} = \hat{C})$: the client computes the tuple $\langle \mathsf{c}^{\mathsf{L}}_x, \mathsf{c}^{\mathsf{R}}_x \rangle$ such that $\mathsf{c}^{\mathsf{L}}_x \leftarrow \mathsf{ORE.Encrypt}_{\mathsf{L}}(\mathsf{sk}, x)$ and $\mathsf{c}^{\mathsf{R}}_x \xleftarrow{\$} \mathsf{ORE.Encrypt}_{\mathsf{R}}(\mathsf{sk}, x)$, and sends it to the server. The server uses $\mathsf{c}^{\mathsf{L}}_x$ to find an insertion position i within $\hat{C}$ such that $\mathsf{ORE.Compare}(\mathsf{c}^{\mathsf{L}}_x, \mathsf{c}^{\mathsf{R}}_i) = \{-1 \vee 0\}$ and $\mathsf{ORE.Compare}(\mathsf{c}^{\mathsf{L}}_x, \mathsf{c}^{\mathsf{R}}_{i+1}) = \{0 \vee 1\}$ (or $i = N$). As for $\mathsf{RQ.Range}$, binary search can be used to compute a logarithmic number of $\mathsf{ORE.Compare}$ operations. Then, the server inserts $\mathsf{c}^{\mathsf{R}}_x$ at the position i within the updated database $\hat{C}'$. The server updates its state to st' as the new database $\hat{C}'$. Clearly, there may be multiple acceptable values of i if the database includes duplicate values.

Limitations of LW-ORE models. The LW-ORE range query protocol models the encrypted database as state information $\mathsf{st} = \hat{C} = \langle \mathsf{c}^{\mathsf{R}}_1, \ldots, \mathsf{c}^{\mathsf{R}}_N \rangle$, which represents the list of *right ciphertexts* sorted with regard to their corresponding plaintexts, that is, $\mathsf{p}_n \leq \mathsf{p}_{n+1}\ \forall n \in [N-1]$. The LW-ORE query protocol assumes that the database server maintains the state information sorted throughout *insert* operations and thus that binary search can always be executed on st (see [8, Section 5.2]). Moreover, the original paper claims semantic security against *snapshot offline adversaries* because *right ciphertexts* are semantically secure (see [8, Definition 5.2, Theorem 5.5]). We observe that the database model and security analyses proposed in the original LW-ORE paper have two major limitations:

- claims about semantic security against *snapshot offline adversaries* do not take into account that although *right ciphertexts* are semantically secure, the encrypted database also leaks the partial sorting order with regard their corresponding plaintexts. It is critical to clearly declare this additional leakage because it may open to known attacks (e.g., correlation attacks on ideal ORE [20]). *Offline security* analyses as proposed in the original paper [8] may apply only if *right ciphertexts* are maintained unsorted and thus queried in linear time. Although we point out this gap in the original security claims, in this paper we do not propose improvements

in this regard because we do not focus on theoretical security analyses. We leave improvements in this regard as future work.

- the database model does not include indexing data structures, which represent auxiliary information that is necessary for adopting the scheme in quite any real database system. As also stated by [19], a *snapshot offline adversary* that operates ciphertext-only inference attacks is able to access to the "steady state" of an encrypted database including all auxiliary information that is needed to perform encrypted searches efficiently (as also reminded in [8, see *Robustness against offline inference attacks*]). Thus, a proper database model should also include indexing data structures. Indeed, real-world examples of tentative LW-ORE implementations adopted standard indexing techniques, such as AVL trees [29] and B+trees [30]. However, we show that both implementations leak additional information with regard to the security claims of the LW-ORE scheme. In the following Section 3, we show necessary conditions for designing secure indexing data structures for LW-ORE, that is, that allow to avoid leaking additional information.

3. Information leakage of standard indexes

We analyze indexing solutions for LW-ORE databases. In Section 3.1 we extend notation proposed by the original LW-ORE paper (see Section 2 and [8]). In Section 3.2 we discuss information leakage due to duplicate values. In Section 3.3 we discuss information leakage related to transaction history.

3.1. LW-ORE document databases with indexes

We extend notation of the original LW-ORE range query protocol (see Section 2) to propose a model that could fit an indexed document-oriented database encrypted with LW-ORE. To this aim, we model the database *state information* as $\mathsf{st} = \langle D, I \rangle$. We denote as $D = \{\mathsf{id} : \mathsf{d}_{\mathsf{id}}\}$ the set of documents possibly encrypted via a standard symmetric scheme, where d_{id} denotes a document uniquely indexed by an opaque document identifier id. We denote as $I = \langle \delta, \{\mathsf{c}^{\mathsf{R}}\}, \{\mathsf{id}\} \rangle$ the indexing data structure that enables efficient range queries on documents D. The links of the indexing data structure are denoted as δ, the *keys* used to evaluate queries are the *right ciphertexts* $\{\mathsf{c}^{\mathsf{R}}\}$, and the pointers to the documents are document identifiers $\{\mathsf{id}\}$.

We analyze how an adversary may try to obtain information from the state of the database. Since both documents $\{\mathsf{d}_{\mathsf{id}}\}$ and *right ciphertexts* are encrypted with semantically secure schemes, an adversary cannot infer any information except possibly for their size and number. Moreover, we assume that document identifiers do not reveal any information (e.g., they are random strings). However, an adversary may try to gain information from the data structure δ. We note that the original paper of LW-ORE did not clearly model this information (see Section 2). In the remaining of the section, we consider different kinds of popular data structures and analyze their security guarantees.

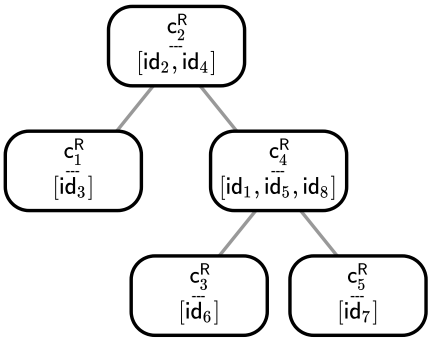


Figure 1: Example of AVL tree for indexing a LW-ORE database.

3.2. Leakage related to duplicate values

One of the main benefits of LW-ORE is to store semantically secure *right ciphertexts* within the database (see Section 2). However, the typical design choice of any indexing data structure is to store all pointers to duplicate database values within the same node to improve query performance and to reduce the index size. The result is that applying standard indexes to LW-ORE leaks *duplicate values* within the database, and thus the security guarantees of LW-ORE fall back to those of a deterministic encryption scheme, possibly re-enabling related inference attacks [19].

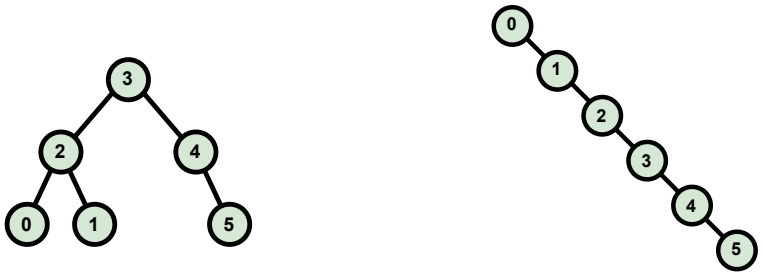
To better explain this class of vulnerability, we consider an example based on an AVL tree used as the indexing data structure for a LW-ORE database, which represents the implementation of [29]. We remind that each node of an AVL tree includes a *key* that is used for traversing the tree, one or multiple pointers to all the *documents* within the database associated with the key, and points to a maximum of two children nodes. The association of multiple pointers to the same key is not an issue for unencrypted databases, but introduces information leakage about *documents* sharing the same *key* values.

A visual example of an AVL tree instantiated for LW-ORE is shown in Figure 1. Recalling notation described in Section 3.1, the *keys* are *right ciphertexts* $\{\mathsf{c}_1^{\mathsf{R}}, \ldots, \mathsf{c}_5^{\mathsf{R}}\}$, and the pointers to *documents* are identifiers $\{\mathsf{id}_1, \ldots, \mathsf{id}_8\}$. It is clear that *snapshot offline adversaries* that access the AVL tree know that documents associated with id_2 and id_4 (id_1, id_5 and id_8) are associated with the same *key*, even if they are not able to compute the $\mathsf{ORE.Compare}$ function.

In Section 4.1 we outline how to design data structures that leak no information about duplicates.

3.3. Leakage related to transactions history

The structure of typical indexing data structure depends on the history of the transactions executed on the database. As an example, even if two databases store the same values, the topologies of their indexing data structures may differ if the insertion order of the values is not the same. Thus, the indexing data structure may leak information about the transactions history. Literature identified this security risk and denoted as *history*



(a) Insertion history $SEQ_A = \{3, 2, 4, 0, 1, 5\}$ (b) Insertion history $SEQ_B = \{0, 1, 2, 3, 4, 5\}$

Figure 2: Example of two plaintext BSTs with different insertion histories.

independent the data structures whose topology and internal bit representation leak no information about the transaction history [32, 33]. In particular, we are interested in *weakly history independent* data structures, which guarantee history independence against adversaries which only observe the indexing data structure once [33], thus fitting the same *snapshot offline adversaries* considered by offline security guarantees of LW-ORE. The adversary that obtains a snapshot of a standard (*history dependent*) data structure may infer information about the transaction history, potentially also revealing information about the distribution of plaintexts if such distribution is not independent of the transaction history.

To clarify the type of information that a *history dependent* data structure may leak, we consider the example represented in Figure 2. We consider two Binary Search Trees (BSTs), both generated by providing the same set of values $\{0, \ldots, 5\}$ in different orders. The first BST (Figure 2a) is built from the insertion sequence $SEQ_A = \{3, 2, 4, 0, 1, 5\}$. The second BST (Figure 2b) is built from insertion sequence $SEQ_B = \{0, 1, 2, 3, 4, 5\}$. The structure of the second BST leaks that data has been inserted within the database in ascending order.

Re-balancing the BST may help hiding information leakage related to the structure topology, however frequent re-balancing may critically affect performance and information may still leak from the internal bit representation [34]. Moreover, although history independent data structures have been designed for efficient range queries [35], they cannot be straightforwardly adopted to secure LW-ORE databases because they may leak duplicate values (see Section 3.2). In Section 4.2 we outline how to design secure history independent data structures that leak no information about duplicates.

4. Designing secure indexes for ORE

We discuss how to design data structures that do not leak information. In Section 4.1 we describe alternative designs for search trees that avoid leaking information related to duplicate values, but that are still history dependent. In Section 4.2 we describe a history independent skip list without duplicates leakage.

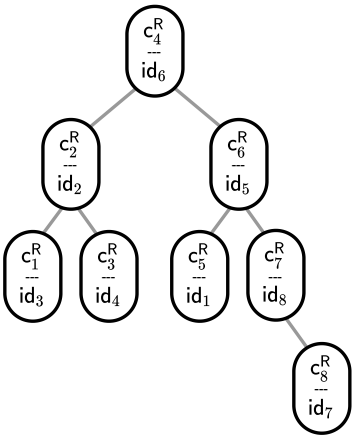


Figure 3: Example of variant AVL tree to avoid duplicates leakage.

4.1. Search tree without duplicates leakage

The duplicates leakage arises when a search tree stores two or more documents that are associated with same *key* (we may informally denote them as *duplicate documents*). As discussed in Section 3.2, standard search trees index *duplicate documents* by storing their identifiers within the same node. Thus, a variant design for search trees that do not leak information duplicates is to store only one identifier within each node. To this aim, it is necessary to design variant algorithms for insertion, search and deletion operations. We propose a candidate design for the AVL tree already considered in Section 3.2. In this paper, we keep the discussion informal and leave improved analyses as a future work.

When indexing a new document by a certain key, the insertion algorithm proceeds unmodified until the same key is found. Then, the algorithm creates a new node as a child of the existing one, deciding whether the new node should be assigned with the left or the right child position randomly with the same probability. If a node already exists at the assigned position, the sub-tree starting from that node is assigned as a child of the new one. When searching for a document, the search algorithm also proceeds unmodified until the target key is found. However, with regard to the original design, the search algorithm cannot assume that all pointers to documents are stored within that node. Instead, the search algorithm must continue descending into left and right sub-trees, collecting all the pointers stored within nodes associated with the same target key, and stopping descending a sub-tree when a node with a different key is found.

We propose an example of the algorithm by referring to Figure 3, which modifies the standard AVL tree described in Section 3.2 (Figure 1).

Although the proposed variant avoids duplicate keys being stored within the same node of the tree, it increases search complexity and thus may affect performance. Moreover, we observe that the variant is still history dependent.

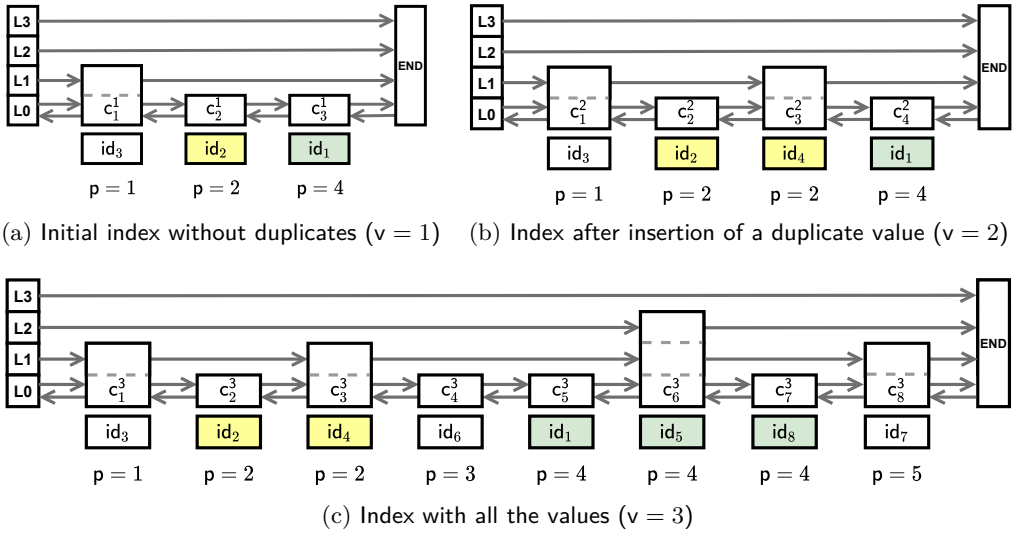
4.2. History-independent skip list without duplicates leakage

We outline how to design an indexing data structure to avoid both information leakage due to duplicates and insertion history. The design is based on skip lists [36], which offer inherently *weakly history-independent* topologies [35], and we denote it as *doubly-linked skip list*. First, we remind the main design traits of skip lists. Then, we describe the proposed variant design.

A skip list is a hierarchical data structure which includes multiple linked lists. At the bottom layer is the *complete list* of all the sorted values stored within the database. Each higher level consists of a *sparse list* which includes a subset of the elements of the list at the level below, and acts as a sort of "fast track" for quicker access. In particular, at insertion time each element has a probability to be *promoted* to the higher layer until promotion fail. Thus, the structure of the *sparse list* is completely independent of the contents of the lists or on the transaction history. The promotion probability p, which is typically set to $\frac{1}{2}$ or $\frac{1}{4}$, determines the average size of the *sparse lists*.

Although the topology of the skip list is *weakly history independent*, it may still leak information about duplicates, because each node of the *complete list* typically stores all the identifiers associated with the same key, as already observed for search trees (see Section 3.2). For this reason, we propose a variant that we denote as *doubly-linked skip list*. Our design adopts the same unmodified superimposed *sparse lists* of a standard skip list. However, we operate two modifications: first, to avoid leaking duplicates we let each node of the *complete list* store only one document identifier (as we already detailed for search trees variants, see Section 4.1); second, to guarantee efficient lookup performance we design the *complete list*, which traditionally is a *linked list* with forward pointers only, as a *doubly-linked list* provided with both forward and backward pointers (which gives the name to our data structure).

We provide a better explanation of our design choices by analyzing how the proposed *doubly-linked skip list* evolves throughout insertion operations. To this aim, we consider the example represented in Figure 4, where we show three versions associated with an increasing number of inserted elements. Within the figure we slightly modify our notations and denote as $\mathsf{c}_n^{\mathsf{v}}$ a *right ciphertext* $\mathsf{c}_n^{\mathsf{R}}$ included within the data structure at "version" v, and id_t the document identifier inserted via the t^{th} operation. We also denote as $\mathsf{L0}$ the *complete list* and as $\mathsf{L1}, \ldots, \mathsf{L3}$ the *sparse lists*. At the beginning ($\mathsf{v} = 1$, Figure 4a), we consider a dataset without duplicate values (e.g., corresponding plaintext values are $\hat{P}^1 = \langle 1, 2, 4 \rangle$) which can be built by using an unmodified skip list insertion algorithm with the only difference that the *complete list* $\langle \mathsf{c}_1^1, \mathsf{c}_2^1, \mathsf{c}_3^1 \rangle$ is a doubly-linked list. When a duplicate key is inserted into the data structure (see Figure 4b, where document identifier id_4 is associated with *right ciphertext* c_3^2 corresponding to plaintext $\mathsf{p} = 2$), the insertion algorithm uses the *sparse lists* to access the *complete list* at the latest position that includes a key that is smaller or equal than the given insertion key, as the original skip list insertion algorithm. If a duplicate key is found, the new encrypted key is inserted after its position in the *complete list*. At insertion, the promotion algorithm is executed to decide whether new encrypted key should be added to upper layer sparse lists. In the example, c_3^2 is inserted after c_2^2 and is promoted to *sparse list* $\mathsf{L1}$. Similarly,



(a) Initial index without duplicates (v = 1) (b) Index after insertion of a duplicate value (v = 2)

(c) Index with all the values (v = 3)

Figure 4: Sequence of insertions within our *doubly-linked skip list* for indexing LW-ORE.

in Figure 4c (v = 3) we also insert values $\langle 4, 3, 5, 4 \rangle$ in this order. Now we consider searching value p = 4 within the index at version v = 3. The search algorithm finds the value within L2, and thus access the *complete list* at position 6 (c_6^3). Now, it operates linear searches over the *complete list* both backward and forward until different values are found. Backward search finds c_5^3 and stops at c_4^3, and forward search finds c_7^3 and stops at c_8^3. The search operation would not be able to efficiently find backward duplicate values without adopting a *doubly-linked list* as *complete list*.

The proposed *doubly-linked list* outlines the main ideas for a secure indexing data structure for LW-ORE databases. Clearly, further analyses and improvements should be considered for obtaining good performance in real database systems, which we plan as future work.

5. Conclusions

We analyzed the security guarantees of a state-of-the-art order revealing encryption scheme when deployed in real database systems, and found issues related to information leakage due to gaps within the original security models and analyses. The most severe security issues are related to information leakage due to indexing data structures, including disclosure of duplicate values within the database and context information related to transactions history. These vulnerabilities may allow adversaries accessing a snapshot of the database to break semantic security guarantees claimed by the encryption scheme or, in certain scenarios, to infer information related to the order of the operations executed on the database and related statistical distributions. We described how variant indexing data structures may be modified to prevent both security issues, and outlined a candidate design based on skip lists. We leave a more formal treatment of the proposed analyses

and the design of a complete proof of concept as future work.

Acknowledgments

We thank Mauro Leoncini for discussions on preliminary versions of this work. This work was supported by Doxee SpA in association with the European Regional Development Fund program “Por FESR 2014-2020” of Emilia-Romagna Region.

References

[1] Microsoft, Transparent data encryption, https://learn.microsoft.com/en-us/sql/relational-databases/security/encryption/transparent-data-encryption, Visited Feb. 2023.

[2] H. Hacigümüş, B. Iyer, C. Li, S. Mehrotra, Executing SQL over encrypted data in the database-service-provider model, in: Proc. ACM SIGMOD Int’l Conf. Management of Data, 2002.

[3] H. Hacigümüş, B. Iyer, S. Mehrotra, Providing database as a service, in: Proc. 18th IEEE Int’l Conf. Data Engineering, 2002.

[4] H. Saleem, M. Naveed, SoK: Anatomy of Data Breaches, in: Proc. 20th Privacy Enhancing Technologies Symp., 2020.

[5] B. Fuller, M. Varia, A. Yerukhimovich, E. Shen, A. Hamlin, V. Gadepally, R. Shay, J. D. Mitchell, R. K. Cunningham, SoK: Cryptographically Protected Database Search, in: Proc. IEEE Symp. Security and Privacy, 2017.

[6] C. Gentry, Fully homomorphic encryption using ideal lattices, in: Proc. 41st ACM Symp. Theory of Computing, 2009.

[7] A. Michalas, The lord of the shares: Combining attribute-based encryption and searchable encryption for flexible data sharing, in: Proc. 34th ACM SIGAPP Simp. Applied Computing, 2019.

[8] K. Lewi, D. J. Wu, Order-revealing encryption: New constructions, applications, and lower bounds, in: Proc. 23rd ACM SIGSAC Conf. Computer and Communications Security, 2016.

[9] E. Damiani, S. D. C. Vimercati, S. Jajodia, S. Paraboschi, P. Samarati, Balancing confidentiality and efficiency in untrusted relational DBMSs, in: Proc. 10th ACM Conf. Computer and Communications Security, 2003.

[10] M. Chase, S. Kamara, Structured encryption and controlled disclosure, in: Proc. IACR 16th Int’l Conf. Theory and Application of Cryptology and Information Security (ASIACRYPT), 2010.

[11] S. D. C. Di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, P. Samarati, Over-encryption: Management of access control evolution on outsourced data, in: Proc. 33rd Int’l Conf. Very Large Data Bases, 2007.

[12] L. Ferretti, M. Colajanni, M. Marchetti, Access control enforcement on query-aware encrypted cloud databases, in: Proc. 5th IEEE Int’l Conf. Cloud Computing Technology and Science, 2013.

[13] M. A. Hadavi, E. Damiani, R. Jalili, S. Cimato, Z. Ganjei, AS5: A secure searchable secret sharing scheme for privacy preserving database outsourcing, in: Data Privacy Management and Autonomous Spontaneous Security: 7th Int'l Workshop DPM 2012, and 5th Int'l Workshop SETOP 2012, 2013.
[14] R. A. Popa, C. M. Redfield, N. Zeldovich, H. Balakrishnan, CryptDB: Protecting confidentiality with encrypted query processing, in: Proc. 23rd ACM SIGOPS Symp. Operating Systems Principles, 2011.
[15] S. Tu, M. F. Kaashoek, S. Madden, N. Zeldovich, Processing analytical queries over encrypted data, in: Proc. Int'l Conf. Very Large Data Bases, 2013.
[16] L. Ferretti, M. Colajanni, M. Marchetti, Distributed, concurrent, and independent access to encrypted cloud databases, IEEE Trans. Parallel and Distributed Systems (2014).
[17] P. Grubbs, R. McPherson, M. Naveed, T. Ristenpart, V. Shmatikov, Breaking web applications built on top of encrypted data, in: Proc. 23rd ACM SIGSAC Conf. Computer and Communications Security, 2016.
[18] P. Grubbs, K. Sekniqi, V. Bindschaedler, M. Naveed, T. Ristenpart, Leakage-abuse attacks against order-revealing encryption, in: Proc. 2017 IEEE Symp. Security and Privacy, 2017.
[19] M. Naveed, S. Kamara, C. V. Wright, Inference attacks on property-preserving encrypted databases, in: Proc. 22nd ACM SIGSAC Conf. Computer and Communications Security, 2015.
[20] F. B. Durak, T. M. DuBuisson, D. Cash, What else is revealed by order-revealing encryption?, in: Proc. 23rd ACM SIGSAC Conf. Computer and Communications Security, 2016.
[21] E. M. Kornaropoulos, C. Papamanthou, R. Tamassia, The state of the uniform: Attacks on encrypted databases beyond the uniform query distribution, in: Proc. IEEE Symp. Security and Privacy, 2020.
[22] V. Bindschaedler, P. Grubbs, D. Cash, T. Ristenpart, V. Shmatikov, The tao of inference in privacy-protected databases, 2018.
[23] S. Oya, F. Kerschbaum, Hiding the access pattern is not enough: Exploiting search pattern leakage in searchable encryption, in: Proc. 30th USENIX Security Symp., 2021.
[24] C. Priebe, K. Vaswani, M. Costa, EnclaveDB: A secure database using SGX, in: Proc. IEEE Symp. Security and Privacy, 2018.
[25] S. Fei, Z. Yan, W. Ding, H. Xie, Security vulnerabilities of SGX and countermeasures: A survey, ACM Computing Surveys (2021).
[26] O. Pandey, Y. Rouselakis, Property preserving symmetric encryption, in: Proc. IACR Advances in Cryptology (EUROCRYPT), 2012.
[27] D. Boneh, K. Lewi, M. Raykova, A. Sahai, M. Zhandry, J. Zimmerman, Semantically Secure Order-Revealing Encryption: Multi-input Functional Encryption Without Obfuscation, in: Proc. IACR Advances in Cryptology (EUROCRYPT), 2015.
[28] N. Chenette, K. Lewi, S. A. Weis, D. J. Wu, Practical order-revealing encryption with limited leakage, in: Proc. IACR Fast Software Encryption, 2016.
[29] P. Alves, D. Aranha, A framework for searching encrypted databases, Journal of

Internet Services and Applications (2018).
[30] D. Bogatov, G. Kollios, L. Reyzin, A Comparative Evaluation of Order-Revealing Encryption Schemes and Secure Range-Query Protocols, in: Proc. Conf. Very Large Data Bases, 2019.
[31] CipherStash, Order-revealing encryption library, https://github.com/cipherstash/ore.rs, Visited Feb. 2023.
[32] D. Micciancio, Oblivious data structures: applications to cryptography, in: Proc. 29th ACM Symp. Theory of Computing, 1997.
[33] M. Naor, V. Teague, Anti-persistence: History independent data structures, in: Proc. 33rd ACM Symp. Theory of Computing, 2001.
[34] T. Moran, M. Naor, G. Segev, Deterministic history-independent strategies for storing information on write-once memories, in: Int'l Colloquium on Automata, Languages, and Programming, 2007.
[35] M. A. Bender, J. W. Berry, R. Johnson, T. M. Kroeger, S. McCauley, C. A. Phillips, B. Simon, S. Singh, D. Zage, Anti-persistence on persistent storage: History-independent sparse tables and dictionaries, in: Proc. 35th ACM SIGMOD-SIGACT-SIGAI Symp. Principles of Database Systems, 2016.
[36] W. Pugh, Skip lists: a probabilistic alternative to balanced trees, Communications of the ACM (1990).