LOCALIZING UNAUTHORIZED UPDATES IN PUBLISHED MICRO-DATA TABLES THROUGH SECRET ORDER-BASED WATERMARKING

SHUAI LIU

A THESIS

IN

THE CONCORDIA INSTITUTE FOR INFORMATION SYSTEMS ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF APPLIED SCIENCE IN INFORMATION SYSTEMS SECURITY

CONCORDIA UNIVERSITY

MONTRÉAL, QUÉBEC, CANADA

FEBRUARY 2010

Your file Votre référence
ISBN: 978-0-494-67248-8
Our file Notre référence
ISBN: 978-0-494-67248-8

Abstract

Localizing Unauthorized Updates in Published Micro-Data Tables through Secret Order-Based Watermarking

Shuai Liu

The study of micro-data disclosure issue has largely focused on the privacy preservation aspect, whereas the integrity of a published micro-data table has received limited attention. Unauthorized updates to such a table may lead users to believe in misleading data. Traditional cryptographic stamp-based approaches allow users to detect unauthorized updates using credentials issued by the data owner. However, to localize the exact corrupted tuples would require a large number of cryptographic stamps to be stored, leading to prohibitive storage requirements. In this thesis, we explore the fact that tuples in a micro-data table must be stored in a particular order, which has no inherent meaning under the relational model. We propose a series of algorithms for embedding watermarks through reordering the tuples. The embedded watermarks allow users to detect, localize, and restore corrupted tuples with a single secret key issued by the data owner, and no additional storage is required. At the same time, our algorithms also allow for efficient updates by the data owner or legitimate users who know the secret key. The proposed algorithms are implemented and evaluated through experiments with real data.

Acknowledgments

I would like to thank a lots of people who provide help for this thesis.

First of all, I would like to express my sincerely thanks to my supervisor, Dr Lingyu Wang, for his valuable advice, unflinching encouragement and constructive criticism throughout my graduate study and research at Concordia University. His motivation, enthusiasm, and immense knowledge inspire my growth as a master student.

I would like to give gratitude to all my friends, for their friendship, understanding and support.

I also appreciate the help from ENCS faculty, my classmates, and all lab colleagues

Last but not least, the greatest regards to my parents, whose unconditional love help me overcome difficulties in my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The micro-data disclosure problem has attracted much attention lately. Most studies have focused on the privacy preservation aspect, whereas the integrity issue has only received limited attention. When a micro-data table is hosted on distrusted servers, the table may contain corrupt or falsified data caused by unauthorized updates. Users of such a table may be deceived into accepting misleading information, which can have serious consequences.

A similar issue has been extensively studied in a different context, that is, outsourced databases [36, 47, 59, 68, 69]. When querying a database hosted by distrusted service providers, the integrity of the query's result can be verified using cryptographic stamps computed over various hash trees built on the underlying relational table. A user can ensure that all and only the queried tuples will be included in the query result, in the correct order, and in their latest version.

In this thesis, we focus on a different aspect of the integrity issue, that is, the localization of corrupt data. This aspect is particularly relevant to users of published micro-data tables. Unlike outsourced databases where a corrupt query result can, and should, simply be rejected, a micro-data table that fails the integrity check as a whole may still contain a large portion of useable data. Moreover, users of a published micro-data table usually do not have the necessary means to immediately contact the original data owner for obtaining a correct copy of the micro-data table. Therefore, it is important for such users to utilize as much as possible the micro-data table, by localizing and disregarding any corrupt data.

The localization of corrupt data with a fine granularity, such as a tuple, inherently requires additional storage. A straightforward solution is to store in a micro-data table the cryptographic stamp of every tuple such that the integrity of each tuple can be verified independently and consequently corrupt tuples can be localized (note that storing a single stamp computed over the root of a hash tree [44] will not achieve the same purpose). However, the storage requirement of this solution will be of the same order of magnitude as the size of the micro-data table.

Instead of demanding additional storage, our solution makes use of the *free* storage already available from the unused order of tuples in a published micro-data table. Unlike tuples in a (outsourced) database, which are not ordered under the relational model, tuples in a published micro-data table must be stored in a particular order even though the order may be meaningless under the relational model (our solution will not apply if this assumption does not hold). With a reasonably large micro-data table, the amount of storage provided by the order of tuples is clearly sufficient (n tuples can provide $log(n!)$ bits) for our purposes.

Our solution can be regarded as a *fragile watermarking* method. Unlike a robust watermark (whose design goal is to tolerate modifications such that the digital copyright can still be protected [4, 60]), a fragile watermark is designed to be easily destroyable such that its absence will signal unauthorized updates [40]. Closest to our work, a fragile watermarking scheme is applied to relational tables [38]. The order between each pair of tuples is used to embed a watermark computed over a group of tuples. The granularity of localization is thus limited to a group, instead of a tuple. If an unauthorized update corrupts tuples spanning many groups, then a large number of unmodified tuples may be mistakenly determined as unusable.

The main contribution of this thesis is a secret order-based fragile watermarking scheme for localizing corrupt data at the tuple level. Basically, the scheme sorts all the tuples in a micro-data table according to a secret order derived from the keyed hash value of those tuples; an unauthorized update will cause conflicts to the expected secret order and thus be detected. In contrast to most existing methods, the proposed scheme has the following key advantages.

1. *Tuple level-localization:* To our best knowledge, this is the first fragile watermarking scheme that can localize corrupt data at the tuple level.

2. *No additional storage:* Unlike most cryptographic techniques, our scheme does not require any additional storage.

3. *Non-Distortion:* Unlike traditional watermarking schemes that modify data to embed watermarks, our scheme does not cause any distortion to data values.

4. *Blind verification:* A single secret key issued by the data owner is sufficient for legitimate users to localize all corrupt data.

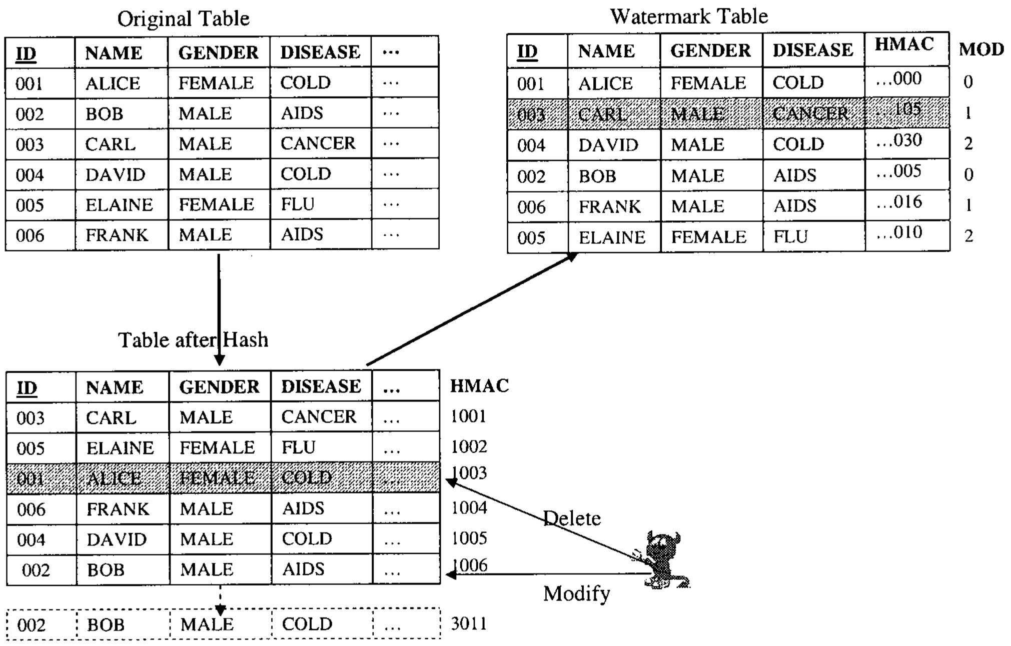
5. *Updatable:* Authorized updates made by legitimate users who know the secret key can be efficiently implemented.

6. *Invisible:* To an adversary who does not know the secret key, the secret order used to embed watermarks is indistinguishable from any arbitrary order.

The rest of the thesis is organized as follows. Section 1.1 builds intuitions through a motivating example. Section 1.2 then states the problem by discussing our assumptions about the system architecture and adversary model. Chapter 2 reviews the literature. Chapter 3 presents the basic scheme for a special case. Chapter 4 extends the basic scheme to address the general case. Chapter 5 evaluates the proposed schemes through extensive experiments. Finally, Section 6 concludes the thesis.

1.1 Motivating Example

A motivating example is depicted in Figure 1. Suppose a data owner would like to share a large number of patients' medical records, which are stored in the format of a relational table shown as the *Original Table*, with researchers at medical institutions and universities for research purposes. Clearly, such data are highly sensitive and cannot be published on the Web. Therefore, the data owner gives each organization a copy of the data, which is to be stored and managed by the latter.



Original Table

ID	NAME	GENDER	DISEASE	···
001	ALICE	FEMALE	COLD	···
002	BOB	MALE	AIDS	···
003	CARL	MALE	CANCER	···
004	DAVID	MALE	COLD	···
005	ELAINE	FEMALE	FLU	···
006	FRANK	MALE	AIDS	···

Watermark Table

ID	NAME	GENDER	DISEASE	HMAC	MOD
001	ALICE	FEMALE	COLD	...000	0
003	CARL	MALE	CANCER	...105	1
004	DAVID	MALE	COLD	...030	2
002	BOB	MALE	AIDS	...005	0
006	FRANK	MALE	AIDS	...016	1
005	ELAINE	FEMALE	FLU	...010	2

Table after Hash

ID	NAME	GENDER	DISEASE	...	HMAC
003	CARL	MALE	CANCER	...	1001
005	ELAINE	FEMALE	FLU	...	1002
001	ALICE	FEMALE	COLD	...	1003
006	FRANK	MALE	AIDS	...	1004
004	DAVID	MALE	COLD	...	1005
002	BOB	MALE	AIDS	...	1006
002	BOB	MALE	COLD	...	3011

Figure 1: An Example of Watermarking Micro-Data Tables

For users of such data to verify that the copy of data has not been subject to unauthorized modifications, a typical solution is to compute a cryptographic signature over the collection of data using the data owner's private key, such that users may verify the integrity of the data using the data owner's public key. However, if the verification fails, the user will not be able to tell which part of the data has been modified and which part is still intact. For the user to notify the organization about such an incident and to further repair the corrupt data either from backups, or by contacting the data owner will apparently take a long time, and in some cases may not be feasible at all. On the other hand, it is desirable for the user to *localize* any corrupt portion of the data, such that the remaining data can still be used.

Consider a simple approach to address the above issue. That is, we compute the keyed hash (for example, the keyed-Hash Message Authentication Code (HMAC)) of each tuple, and re-order these tuples using the hash values as indices. Assume a perfect situation where all the hash values happen to be distinct and continuous, which is shown as the *Table after Hash* in Figure 1. An authorized user who knows the cryptographic key can recompute the hash value of each tuple, and check whether these tuples are sorted continuously according to those recomputed hash values. On the other hand, for example, if an intruder or unauthorized user may modify Bob's record, then this tuple's hash value may change to, say, 3011; if he/she deletes Alice's record, the remaining tuples' hash values will not be continuous. Therefore, the attacks can be detected and localized in either case.

Unfortunately, the assumption that all hash values will be distinct and continuous is clearly unrealistic. Indeed, even a perfect hash function will give outputs that are uniformly distributed, but only statistically. In practice, when the hash values are sorted, we may find many duplicates or gaps between values that prevent them to be used as indices of tuples. It is worth noting that although we can still sort tuples based on such hash values in, say, a non-decreasing order, the capability of localizing modified tuples will be reduced, since we will not know the exact hash value to be expected for each tuple.

Therefore, instead of directly using each hash value h as the index for the tuple, we use its modulo $h \ mod \ n$, where n is a chosen modulus known to the world. Since all the tuples will now yield a value in $[0, n-1]$, we can arrange tuples based on such values in groups of size n. The result is shown in Figure 1 as the *Watermark Table*. Although this modulo operation will not completely eliminate the above issue, it will increase the chance of success. More precisely, the less n is, a larger percentage of tuples can produce distinct and continuous indices, although for a smaller group (in an extreme case, when $n = 1$, each tuple forms its own group). However, since a modified tuple has $1/n$ probability to yield the same result, a smaller n also means less capability of detecting an unauthorized modification. In this thesis, we will study how to choose the value of n and how to deal with the tuples that cannot be arranged in any complete group.

1.2 Assumptions

We consider the micro-data disclosure application illustrated in Figure 2. There are three parties involved. A *data owner* would like to publish micro-data in the form of a relational table. The table will be duplicated and hosted at a *server* and accessed by *users*. In this architecture, the communications between data owner and server, as well as server and authorized users are assumed to be reliable. However, the server is semi-trusted in the sense that it may be accessed by unauthorized users who may alter the micro-data tables stored on the server (this situation is similar to that of a outsourced database [26]). Therefore, users should verify the integrity of tables stored on the server.

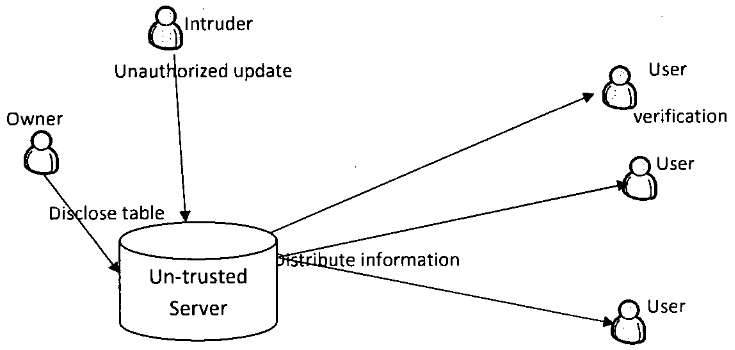


Figure 2: The Architecture

We assume the data owner shares a secret key with each legitimate user of the micro-data. This secret key will enable such those users to verify the integrity of the published table and localize corrupt data, if any. In addition, a user knowing the secret key can also make a *legitimate update* to the published table (for example, an administrator of the server may need to restore some corrupt data from time to time). However, except the shared secret key, we assume there is no interaction between the data owner and the users. Therefore, when a user detects the published table to be corrupt, he/she cannot easily obtain a valid copy of the table. Instead, he/she must try to utilize as much as possible the table by localizing and disregarding the corrupt tuples.

We assume that the watermark embedding and verification methods and parameters such as the modulo value are all known to the world. Besides, as the server is semi-trusted, an intruder can do any kinds of attack on the data, such as insertion, deletion or modification of any tuple. We shall study the case where authorized users may know each attack's type through, for example, system log. We shall also consider cases where the type of each attack is unknown. For different cases, the verification of watermarking will be different, as we shall discuss later.

We assume the adversary model that an intruder's goal is to deceive users into accepting modified micro-data. Naturally, the intruder will attempt to repeat an attack many times so even one of them is not detected, the attack is still a success. Therefore, in our model, we assume a policy that if the number of modified tuples localized by a user exceed a threshold only known to the user, the user will simply stop using the whole table. Such a policy will discourage an intruder from repeating his attacks for many times, since too many repeated attacks will more likely cause the user to discard the whole table, which does not help the attacker in achieving his/her goal. As our scheme is only an integrity mechanism, it cannot prevent a denial of service attack with the goal of simply destroying all the data.

The main challenges of this work are as follows.

Providing fine-grained localization capability without additional storage requirements

Unlike traditional cryptographic methods or fragile watermarking schemes whose goal is usually to detect, instead of localizing modifications, our scheme aims at fine-grained, tuple-level in particular, localization of modifications. Although this can be achieved by, say, creating a cryptographic stamp for each tuple separately, the storage requirements would be prohibitive. Ideally, the amount for storage required by integrity mechanisms should be kept as low as a constant compared to the size of database. It is a challenging problem to meet these seemingly conflicting goals. The way we tackle this issue is to embed watermarks in the unused order of tuples.

Ensuring the accuracy of localization with unreliable storage The accuracy of localization, more precisely the rate of false positives and false negatives, will critically depend on the watermarks that have been embedded. However, since such watermarks are part of the micro-data (that is, the order of tuples), any attack may destroy the watermarks. How to ensure reasonably low false positives and negatives with destroyable watermarks is a challenging problem. We tackle this issue through a majority voting scheme in recovering embedded cryptographic information.

Localizing unauthorized modification while allowing legitimate updates Unlike traditional fragile watermarking schemes which must be applied again to the entire database in order to re-build any corrupted watermarks, our goal is to allow legitimate updates to be carried out efficiently, without having to rebuild all the watermarks. This is challenging since the two goals are seemingly conflicting. We devise efficient updating schemes for this purpose.

Chapter 2

Review of Literature

We review the state of the art in outsourced databases, micro-data disclosure, and watermarking in databases, and so on.

2.1 Outsourced Databases

The rapid growth of digital data and storage management costs requires finding new effective methods to manage data. One of such new paradigm is database outsourcing, which employs a remote server that serves as storage for data owner to distribute their data. In [26], Hacigumus et al build a data management model, which includes three parts: the data owner, the database service provider (server) and the user. Users can access and query data from server instead of data owner. This model satisfies the needs of effective data processing while the traditional database is of higher cost in hardware and management. However, as the servers may be untrusted or compromised, the security concerns are taken into consideration, such as the data privacy and query integrity [54].

Many of those work concentrate on data privacy [26] [5] [25]. The basic idea for protecting privacy is to store encrypted data in server's side instead of storing the original data. There are many methods for users to decrypt data [22] [10]. Chor [10] et al work on how users can retrieve data from outsourced server. Besides, Song et al [61] bring in a searching algorithm for encrypted data to satisfy the requirement of hidden queries.

On the other hand, the integrity in outsourced database just attracts limited attentions in recent years. In query integrity, the returned query results from server must be correct, complete and fresh [36]. The integrity verification methods are usually based on the signature and Merkel Hash Tree [44] [46]. In the signature based techniques, the data owner signs each record in database [48] [50].

Xie et al [68] propose an approach for protecting integrity based on probabilistic methods [68]. In this method, the data owner inserts a small amount of records into outsourced databases which can be tracked by client in query results to make sure the integrity is intact. However, this signature-based methods result in additional storage overhead in server's side. Besides, the signatures' computation time also mean a high cost for data owners.

The techniques of using Merkel Hash Tree to verify integrity, with less storage and computation cost, are popular for integrity protection [16] [51] [9] [47]. The Merkel Hash Tree is a binary tree that hierarchically organizes tuples' digests. Each leaf node contains the digest of a record while each internal node contains the concatenation of the children digests. The root is signed by data owner using the public key system. Given a range query, the server determines boundary records, and returns the verification object (VO) together with the root's signature to user. Then user can use data owner's public key and returned information to verify query results' integrity.

Devanbu et al [16] first using Merkel Hash Tree to authenticate the returned query result's integrity. This method signs each record by RSA [43] scheme and stores them using Merkel Hash Trees. There are others works that study how to verify the completeness of query results [36] [59] [50]. Pang et al [50] use signature aggregation to combine each record's information with other two neighbor records. Therefore, the continuity of query results can be verified using the signature to prove completeness.

Yang et al [71] use authenticated data structures to audit soundness and completeness. Xie et al [69] use an improved authenticated data structure-based approach to guarantee freshness. Mykletun et al [47] compare several schemes based on the minimal computational and bandwidth overhead. However, hash tree structures cannot localize modifications at a fine-grained level but can only detect them. Mouratidis et al [46]introduce a Partially Materialized Digest scheme, which uses separate index for data and verification information with the purpose of reducing the overhead. Besides, there are still some other issues discussed in outsourced database, such as the mechanism [59] and inference issues [12].

2.2 Micro-Data Disclosure

The micro-data disclosure problem has received significant attention lately [1,6,27,33,34]. In particular, data swapping [24,52,62] and cell suppression [35] both aim to protect micro-data released in census tables, but those earlier approaches cannot effectively quantify the degree of privacy. A measurement of information disclosed through tables based on the perfect secrecy notion by Shannon is given in [23]. The authors in [15] address the problem ascribed to the independence assumption made in [23]. The important notion of k-anonymity has been proposed as a model of privacy requirement [56]. It has received tremendous interest in recent years. To achieve optimal k-anonymity with the most data utility is proved to be computationally infeasible [45].

A model based on the intuition of *blending individuals in a crowd* is proposed in [57]. A personalized requirement for anonymity is studied in [70]. In [28], the authors approach the issue from a different perspective, that is, the privacy property is based on generalization of the protected data and could be customized by users. Much efforts have been made around developing efficient k-anonymity algorithms [2,3,18,21,32,55,56], whereas the safety of the algorithms is generally assumed. Many more advanced models are proposed to address limitations of k-anonymity. Many of these focus on the deficiency of allowing insecure groups with a small number of sensitive values, such as l-diversity [42], t-closeness [37], $alpha$-k-anonymity [67], and so on. In addition, a generic model called GBP was proposed to unify the perspective of privacy guarantees in both generalization-based publishing and view-based publishing [14].

While most existing work assume the disclosed generalization to be the only source of information available to an adversary, recent work [73] [66] shows the limitation of such an assumption. In addition to such information, the adversary may also know about the disclosure algorithm. With such extra knowledge, the adversary may deduce more information and finally compromise the privacy property. In the work of [73] [66], the authors discover the above problem and correspondingly introduce models and algorithms to address the issue. However, the method in [66] depends on a specific privacy property, whereas the one in [73] is more general, but it also incurs a high complexity. A special case of the k-jump strategy is discussed in [74] where all jumps end at disclosing nothing.

In contrast to micro-data disclosure, aggregation queries are addressed in statistical databases [27, 30, 49]. The main challenge is to answer aggregation queries without allowing inferences of secret individual values. The auditing methods in [17, 19] solve this problem by checking whether each new query can be safely answered based on a history of previously answered queries. The authors of [19, 29, 31] considered the same problem in more specific settings of offline auditing and online auditing, respectively. The authors of [31] considered knowledge about the decision algorithm itself. However, the solution in [31] only applies to a limited case of aggregation queries and it ignores the current state of the database in determining the safety of a query.

2.3 Watermarking in Databases

Digital watermark is a cryptographic technique to embed information, which can be categorized as fragile watermarks and robust watermarks. For different purposes, we need to design different watermark schemes [11]. Such application as broadcast monitoring, identification, authentication, prove copyright, fingerprint, copy control, and convert communication require the robustness, tamper resistance, fidelity, low cost, and high detection rate. These kinds of watermarks are embedded in image, video and audio [13]. Nowadays the watermarking techniques are being applied to databases. Two kinds of watermarks are adopted in databases, the robust watermarks and the fragile watermarks [40]. Robust watermark is used to protect copyright and owner's identification [60] [39] [4] [72] [63] [53]. Agrawal and Kiernan [4] present a watermarking algorithm for databases to detect piracy. They embed watermarks in relational data and prove the robustness on a wide range of attacks. Later, a zero-knowledge based watermarking scheme [72] [63] [53] was proposed as an asymmetric key method to verify the watermark without disclosing any information. Li et al [39] insert fingerprints in non-primary key attributes to protect copyright. These work have an emphasis on preventing piracy but cannot protect the data integrity. In the last few years, the research on fragile watermarks in multimedia have been conducted for the integrity of images, audio, and video [8] [41] [20]. But the fragile watermarks in databases still have not been adequately addressed. Li et al [38] propose a new technology to insert invisible signals to localize modifications without distortions, using fragile watermarks. In that paper, the watermark cannot provide a small detection and localization granularity. Therefore, we propose new methods to embed fragile watermarks. Unlike other steganography algorithms, our method does not insert any information to databases, but using tuples' order to localize modifications.

Chapter 3

A Special Case

In this chapter, we study a special case where all the keyed hash values modulo n can be divided into sequences with those within each sequence being distinct and continuous. Although this case is unrealistic in practice, it facilitates the introduction of our basic scheme. We will study the general case in next chapter. We present the scheme for the special case in two steps, namely, the embedding and verification of watermarks, respectively.

3.1 Watermark Embedding

Our basic scheme for watermark embedding is a symmetric key system that works on all tuples in a micro-data table. Given a micro-data table T, which is denoted in this thesis as a sequence of m tuples $T = (t_0, t_1, \ldots, t_{m-1})$. Denote by $h(k_h, t)$ a keyed hash function that takes the inputs of a tuple t a secret key k_h which is shared between the data owner and user. Table 1 summarizes the notations that will be used in this thesis.

Definition 1 *Given a micro-data table T with $|T| = m$ and a chosen modulus $n \leq m$, define a function $sid(.) : T \rightarrow [0, n-1]$ as $sid(t) = (h(k_h, t))$ mod n. We say $sid(t)$ is the* sequence index *of tuple t.*

m	number of tuples in original table T
n,n_i	module value, module value in i^{th} level
k_h	the key for key-hash function
k_p	the permutation key
T	the original table
W	owner's watermark table
M	two dimensional array
$sid(.)$	sequence index (function)
$gid(.)$	group index (function)
$seq(.)$	watermark sequence (function)
$grp(.)$	group (function)

Table 1: Notations

We use this function $sid(.)$ to compute each tuple's sequence index, which is essentially the tuple's relative position inside a sequence of n tuples sorted based on their sequence indices. We will show how to decide which sequence each tuple should be placed in shortly.

Definition 2 *Define a function* $grp(.) : [0, n-1] \rightarrow 2^T$ *as* $grp(x) = \{t : t \in T, sid(t) = x\}$*, where* x *is the sequence index.*

We need the function $grp(x)$ to obtain the set of tuples all with the same sequence index x. We shall also abuse the notation $grp(.)$ for the sequence obtained by sorting its tuples based on the order of their appearances in T, namely, a *group*.

Definition 3 *Define a function* $gid(.) : T \rightarrow [0, n-1]$ *as* $gid(t_i) = | \{t_j : t_j \in grp(h(k_h, t_i)) \wedge j < i\} |$. *We say* $gid(t_i)$ *is the* group index *of tuple* t_i.

We use the function $gid(.)$ to obtain each tuple's group index, which means the tuple's relative position inside its group. Notice the difference between a tuples's position inside a sequence and that inside a group. The former is decided by the keyed hash function while the latter by how it is ordered with other tuples with the same key hash value.

Definition 4 *Define a function* $seq(.) : [0, n-1] \rightarrow 2^T$ *as* $seq(y) = \{t : t \in T, gid(t) = y\}$*, where* y *is the group index.*

We use this function $seq(y)$ to get the set of tuples with the same group index y, we shall also abuse $seq(.)$ for the sequence obtained by sorting its tuples in the ascendingly order of their sequence indices, namely, a *watermark sequence.* If $\mid seq(y) \mid = n$, we say $seq(y)$ is a *complete watermark sequence.*

Definition 5 *Define a function* $mas(.) : [0, n-1] \rightarrow [0, n-1]$ *as* $mas(x) = \mid \{x' : (\mid grp(x') \mid > \mid grp(x) \mid) \vee (\mid grp(x') \mid = \mid grp(x) \mid \wedge x \leq x')\} \mid$. *We call the sequence* $mas(0), mas(1), \ldots, mas(n-1)$ *the* master order.

We use this function $mas(.)$ to permute each sequence index x to a new value y in $[0, n-1]$ based on the size of $grp(x)$, in order to make all the group's size in the ascendingly order. That is: $\mid grp(mas^{-1}(0)) \mid \geq \mid grp(mas^{-1}(1)) \ldots \geq \mid grp(mas^{-1}(n-1)) \mid$. Such a property will be useful when we deal with incomplete watermark sequences later in this thesis. Also, from now on, when we mention sequence index, we refer to the sequence index after the permutation based on $mas(.)$.

Definition 6 *Denote by M a two dimensional array that stores all the tuples in T according to their sequence indices and group indices. Specifically, tuple t will become the array element $M[gid(t)][mas(sid(t))]$.*

For the time being, we shall only consider a special case where the m given tuples' sequence indices and group indices will yield only complete sequences of n tuples. In another word, we have n groups with exactly the same size m/n. Therefore, we can place each element of array M, which is a tuple, into the watermark table W first by each row, then by each column.

However, as the algorithm and the modulus n are both public, unauthorized users can also calculate all tuples' sequence indices and group indices in W. This knowledge may lead to targeted attacks that may go undetectable, such as deleting a complete sequence, or swap tuples with the same sequence index, and so on. We thus introduce another pass of permutation of tuples on the watermarked table W to randomize the watermarks, that is, the order of tuples.

Definition 7 *Define a permutation function $per(W, k_p)$ that randomly disturbs all tuples' order in the watermark table W such that the watermark cannot be easily identified from the final disclosed table without knowing the secret key.*

We use function $per(.)$ to re-order the tuples in table W into a new order that is generated based on the secret key k_p, which can be derived from k_h using a one-way function, so only authorized users can recover the watermark table W from the disclosed table. In this thesis, we will use the Lexicographical Order Generation scheme [58] as the permutation function.

grp(mas^{-1}(0))	grp(mas^{-1}(1))			grp(mas^{-1}(n-1))	
M[0][0]	M[0][1]	M[0][2]	......	M[0][n-1]	seq(0)
M[1][0]	M[1][1]	M[1][2]	......	M[1][n-1]	seq(1)
M[2][0]	M[2][1]	M[2][2]	......	M[2][n-1]	seq(2)
......	......	......	......	......	
......	......	......	......	......	seq(|grp(mas^{-1}(n-1))|-1)

Figure 3: Watermark Embedding in the Special Case

Figure 3 illustrates an example of the two dimensional array M. Each row in M is a watermark sequence; each column is a group; each element is a tuple. By definition, we have the following property:

$$| grp(mas^{-1}(0)) \geq | grp(mas^{-1}(1)) | \geq \ldots \geq | grp(mas^{-1}(n-1)) |$$

The size of M is thus:

$$| grp((mas^{-1}(0))) | *n$$

The number of complete watermark sequences is

$$\mid grp(mas^{-1}(n-1)) \mid$$

In this special case, we have that

$$\mid grp(mas^{-1}(0)) \mid = \mid grp(mas^{-1}(n-1)) \mid = m/n$$

Algorithm 1 Algorithm *Emd_Complete*

```
Input: Original table $T$, module $n$, key $k_h$ and $k_p$
Output: Table to be disclosed
Method:
1. For $i = 0$ to $m - 1$
2.     Let $M[gid(T[i])][mas(sid(T[i]))] = T[i]$
3. For $i = 0$ to $\mid grp(mas^{-1}(n-1)) \mid * n - 1$
4.     Let $W[i] = M[\lfloor i/n \rfloor][i \ mod \ n]$
5. Return $per(W, k_p)$
```

Algorithm 1 describes the watermark embedding algorithm in the special case. In this case,$\mid grp(mas^{-1}(0)) = \mid grp(mas^{-1}(1)) \mid = \ldots = \mid grp(mas^{-1}(n-1)) \mid$, so all the tuples are in complete sequences. Line 1 and 2 compute tuples' sequence indices and group indices, and place the tuples into a two dimensional array M. Line 3 and 4 picks tuples from M and put them into watermark table W. Line 5 permutes table M by key k_p, and publish the table. The time complexity for embedding algorithm is $O(m)$, depending on the micro-data table's size m.

3.2 Watermark Verification

The verification of data integrity usually has two aspects: detection and localization. The objective of detection is to decide whether the table is tampered as a whole, whereas the objective of localization is to locate the attacked tuples' positions and types. Notice that although these are for different practical purposes, the former is essentially the coarsest-grained case of the latter. Due to this fact, in our watermark verification method, we combine the detection and localization as two steps of a single process.

Definition 8 *The expected sequence index of a tuple t is the one calculated based on the relative position that t appears in the disclosed table by assuming there is no unauthorized modifications.*

In the special case, as users know the modulus n and the embedding algorithm, they can calculate the expected sequence indices as $I = (0, 1, \ldots, n-1, 0, 1, \ldots, n-1, \ldots)$. The expected sequence indices are then compared with the actual sequence indices computed from the actual tuples in the disclosed table, to check whether the two match. If the two sequence index mismatches for a tuple, then we mark this tuple as suspicious. If all tuples in the received micro-data table have their sequence indices match, we consider this table intact. Otherwise, the table is considered tampered (detection) and we need to localize the attacks.

In the situation that only one known type of attack occurs in the watermark table (for example, the types can be extracted from log files), suppose an intruder modifies some tuples in the table. We can simply localize all mismatched tuples as being attacked. However, if an intruder inserts or deletes some tuples, then the remaining tuples' expected sequence indices will all be affected. So the mismatched tuples are not necessarily attached. Moreover, in the case that attacks' types are unknown or mixed in an unknown manner (which we call combination attack), then we cannot simply regard the mismatched tuples as attacked ones, either. Based on the assumption that attackers' objective is stealthy attacks (instead of denial of services), the least number of attacks that lead to all the mismatches in the disclosed table will be the most plausible case. Therefore, we shall apply the edit distance concept to localize attacks. In general, the edit distance between two strings refers to the operations that one string needs to be transformed into another string [64].

Definition 9 *In the verification of watermarks, by regarding the expected tuples' sequence indices and actual tuples' sequence indices as two character strings, a minimal set of attacks that can transform the former into the latter is referred to as the edit distance.*

There exist different notions for edit distance. In this thesis, we shall use the Levenshtein distance [64]. Levenshtein distance comes with a method for finding the minimal set of operations transforming between two strings. In our scheme, as we mentioned the previous sections, users will adopt a policy that lead them to disregard the whole table when the number of localized attacks exceeds a threshold. Therefore, the intruder's strategy would be to minimize the amount of attacks and the edit distance method thus is suitable for our scheme. By calculating the minimal edit distance, we can obtain the set of attacked tuples. Such a method is suitable for both known types of attacks and combination attack.

Algorithm 2 describes the watermark verification algorithm in the special case. A user receives a disclosed table U and needs to verify its integrity and localize attacks, if any. Line 1 inverts the permutation $per(.)$ to obtain the watermark table V from W with the key k_p (derived from k_h). After the user obtains V, he/she can calculate the number of complete sequences q in V simply by diving the total cardinality by n. Line 2 and 3 compute the actual sequence indices in table V and place them in sequence L. Line 4 and 5 compute the expected sequence indices for table V, and place the results in sequence I.

Algorithm 2 Algorithm $Ver\_Complete$

```
Input: Disclosed table $U$, module $n$, key $k_h$ and $k_p$
Output: Attacked tuples in $U$
Method:
1.  $V = per^{-1}(U, k_p)$ // $V$ is the watermark table
2.  For $i = 0$ to $q * n - 1$ // $q$ is the number of complete sequences in $V$
3.      Let $L[i] = mas(sid(V[i]))$
4.  For $i = 0$ to $q * n - 1$
5.      Let $I[i] = i \ mod \ n$
6.  Return $LevenshteinDistance(I, L)$
```

In the special case, each tuple's position is decided by its sequence index and group index, so in table V each tuple $V[i]$ has the expected sequence index $i \ mod \ n$. Line 6 then applies the standard edit distance algorithm to the actual sequence indices L and expected sequence indices I to obtain the set of attacked tuples, and the result also contains attacks' types. If the table U's integrity is intact, the algorithm will return an empty set. When we use Levenshtein distance to locates attacks' set, the worst-case complexity is $O(m^2)$, so the verification algorithm's time complexity is $O(m^2)$.

3.3 Detection Probability

Now we estimate the probability of successfully detecting an altered table. We shall study only the case of known types of attacks, the case of combination attack and the accuracy of localizing attached tuples will be studied later through experiments. Suppose the intruder has attached l tuples. Due to the users' threshold-based policy, l can be assumed as a small number compared to the total number of tuples.

Due to the extra pass of permutation on watermarked table before the result is disclosed, it would be computationally infeasible for any intruder to know where adjacent tuples in the watermarked table will be placed in the disclosed table. Therefore, it is reasonable to assume that intruders cannot pick which tuples to attack based on the watermarked table. In another word, the attacked tuples will be almost uniformly distributed in the watermarked table, no matter how intruders choose tuples to attack.

A modification attack will alter a tuple's hash value, and consequently sequence index and its position in the watermarked table. An insertion or deletion attack will change many tuples' sequence indices and their positions in the watermarked table. Therefore, these attacks should normally be easily detected. However, there do exist special cases where the attacks cannot be detected, as we shall discuss in the following.

Modification: If a tuple is modified but the new tampered tuple yields the same sequence index as the original one does, then this attack cannot be detected. In our scheme, each tuple's sequence index is in the domain $[0, \ldots, n-1]$ so the tampered tuple has $1/n$ probability of not being detected. Since the detection fails if all l modified tuples have their sequence indices unchanged, the detection probability is

$$P_{dec} = 1 - (1/n)^l$$

. Notice that since the intruder does not know the secret key, he/she has no means to know what is the expected hash value of each tuple so this detection probability cannot be reduced through brute force attacks.

Deletion: When intruder deletes one tuple, the remaining tuples' expected sequence indices will all be changed. So the user can easily detect the attack. However, if the intruder deletes complete sequences, then the deletion cannot be detected. As the modulus n is public, the intruder can choose l as a multiple of n. Now we compute the detection probability. The deleted attacks are randomly distributed in watermark table due to the extra pass of permutation of tuples in the watermarked table. Since the intruder would like to delete l/n complete sequences while there are ⌈m/n⌉ sequences in the table, the detection probability is

$$P_{dec} = 1 - (C^{l/n}_{\lceil m/n \rceil} * C^{n}_{n})/C^{l}_{m}$$

.

Insertion: When the intruder inserts one tuple, the remaining tuples' expected sequence indices will also be changed. Similarly, if the intruder inserts complete sequences, then the insertion cannot be detected. Suppose the intruder chooses l as a multiple of n, and inserts tuples randomly. Suppose after insertion, there are $(m+l)!/m!$ possible results for the tampered table, and only inserting at $\lceil m/n \rceil + 1$ positions can lead to undetected complete sequences. Besides, the probability that the n tuples randomly inserted can form a complete watermarked sequence is $(1/n)^n$. So the probability for detecting insertion attacks is

$$P_{dec} = 1 - (1/n)^n * (C^{l/n}_{\lceil m/n \rceil + 1}/((m + l!/m!))$$

.

For combination attacks or unknown types of attacks, the detection probability will depend on each type of attacks but also on their interaction since different attacks may cancel each other's effect on sequence indices. Also, the localization of attached tuples will depend on specific cases of attacked tuples' position and types of attacks. Therefore, we shall rely on experiments to measure the rate of false positives and false negatives in those cases. Specifically, false positive indicates the case of intact tuples which are mistakenly localized by our verification algorithm as attached tuples, whereas false negative indicates the case of altered tuples which are not localized. The result will be discussed in Chapter 5.

The accuracy of detection depends on the modulus n. A larger n can reduce the number of tuples that yield the same modulo result and leads to a more precise detection result. However, when n is larger, it is usually harder to obtain complete sequences from keyed hash values. Therefore, we should consider the tradeoff between these two aspects, that is, the upper bound of detection probability $1/n$ and the number of incomplete watermark sequences (which will also reduce the actual detection probability). This will be studied through experiments later.

Chapter 4

The General Case

In this section, we discuss the general case where not all the tuples in a given micro-data table can form complete watermark sequences. That is, besides complete sequences, there are still some tuples left whose sequence indices will not allow complete sequences to be formed, namely, the *incomplete part* of the watermarked table. In order to protect those tuples' integrity, we still need to embed watermarks for such an incomplete part of the table. However, we will need different methods, and the capability of detecting and localizing attacked tuples will likely be lower. We study three different algorithms for embedding and verifying watermarks in the incomplete sequences.

4.1 Establishing Master Order

Figure 4 illustrates the two dimensional array M in the general case. In this $\mid grp(mas^{-1}(0)) \mid *n$ array, there are $\mid grp(mas^{-1}(n-1)) \mid$ complete watermark sequences. From the $seq(\mid grp(mas^{-1}(0)) \mid)^{th}$ sequence, the last portion of sequence indices will not correspond to any tuples. That is, we start to have incomplete watermark sequences of size less than n. Now we study how to embed watermarks for those incomplete sequences.

First, we need to discuss the master order function $mas(.)$ for computing the watermark

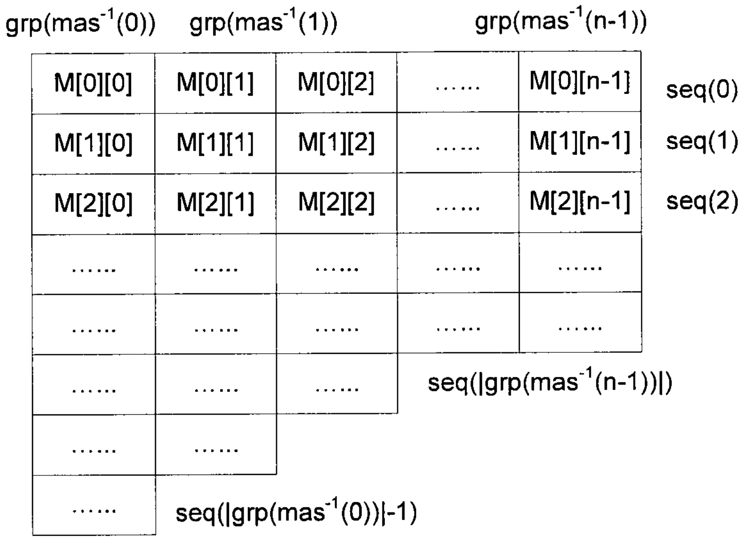


Figure 4: An General Case

table W. The master order function can simply be the identity function (that is, no re-ordering) for the special case. However, in the general case, when all groups have different size, we need $mas(.)$ to keep all the groups' size in descending order. That is, we would like each incomplete sequence to be no shorter than those that appear later.

As we do not require extra storage, and users will hold nothing but a single secret key, the master order function itself must be embedded in the watermarked table using the order of tuples. A straightforward approach is for users to always assume the order of the first complete watermark sequence to be the master order. Compare the order of this first sequence to the indices $(0, 1, \ldots, n-1)$, users can establish the master order function. Due to the extra pass of permutation on the watermarked table, intruders will not be able to know where to locate the tuples of this first watermark sequence, and they thus cannot purposely attack such tuples.

Since the master order is critical to the verification of all tuples, we need its embedding to be more robust against attacks. A simple way to decrease the chance of having the master order destroyed by attacks is to repetitively embed it. More specifically, users may choose the first r complete watermark sequences and taking a majority vote approach to get the most plausible master order. This will increase the chance of having an intact master order, since the intruder must happened to have destroyed a majority of these r sequences before the master order becomes impossible to recover. However, this approach will not work since if one or more tuples are deleted or inserted, then users will not be able to locate the right starting and ending tuples for each of the first r complete watermark sequences. A majority vote will yield wrong result.

Therefore, we adopt a slightly different approach, that is, we reorder the first r complete sequences such that all tuples with the same sequence index will be adjacent to each other, and different sequence indices will still follow the master order. When users extract these sequence indices (may have been modified/deleted/inserted), they will apply majority voting to each r adjacent tuples (which are expected to yield exactly the same sequence index). If any attack is detected in each r adjacent tuples, users will use the information about such attacks (for example, deletion or insertion) to adjust the beginning and ending tuples of the next r adjacent tuples.

For example, suppose the master order is: $(4, 0, 1, 2, 3)$ and $r = 4$. In table W, the first $r * n = 20$ tuples' sequence indices should be: $(4, 4, 4, 4, 0, 0, 0, 0, \ldots, 3)$. In the received watermark table, suppose one tuple in the first 4 tuples is deleted and the result is now: $(4, 4, 4, 0, 0, 0, 0, 1, \ldots, 3)$. User computes first four tuples' sequence indices $(4, 4, 4, 0)$, and get the result 4 by majority voting. As the next tuple's sequence index is still 0, so users can deduce there may be a deletion attack in the first 4 tuples. Users can then attempt to locate the next 4 tuples from the 4^{th} tuple instead of the 5^{th} tuple. Although there are still cases that this method leads to a wrong result, the false positive and negative rate will be reduced significantly, as will be partially reflected in our experiments.

4.2 Algorithm 1

4.2.1 Watermark Embedding

When we place all tuples in T into the two dimensional array M, we would have $\mid grp(mas^{-1}(n-1)) \mid$ complete watermark sequences, which can be directly put into the watermark table W. However, unless in the aforementioned special case, there would be some incomplete watermark sequences remaining in M that also need to be put into W in some way. If we directly put those sequences into W, users would not be able to identify the beginning and ending of each incomplete watermark sequence, since such incomplete watermark sequences will have varying (more precisely, smaller and smaller) sizes. Therefore, we need different watermark embedding algorithms for these incomplete watermark sequences.

The basic idea of algorithm 1 is borrowed from the Lempel-Ziv-Welch lossless data compression algorithm [65]. For each incomplete watermark sequence, we need to indicate the length of the sequence. Due to the master order function, we know that each incomplete watermark sequence will include continuous sequence indices starting from 0. Therefore, for each incomplete watermark sequence, we can simply use the last tuple t as an indicator of the length of this sequence, and place it in front of the sequence, so users would know that the current sequence's indices will start from 0 and end at $mas(sid(t))$.

Algorithm 3 describes the watermark embedding algorithm for this method. Line 1 calls the *Emd_complete* function and get $\mid grp(mas^{-1}(n-1)) \mid$ complete sequences. For each incomplete watermark sequence in M, Line 3 and 4 add length indicator to it by moving the last tuple to the beginning of the sequence. Line 6 and 7 simply place the remaining tuples in the incomplete watermark sequence into table W. Line 9 permutes the table M using the key k_p, and the result is ready to be disclosed. The time complexity for this algorithm is also $O(m)$, depending on the micro-data table's size m.

For example, suppose $n = 10$, for the incomplete sequences, suppose $\mid grp(0) \mid = 10$,

Algorithm 3 Algorithm *Emd_Alg1*

```
Input: Original table $T$, module $n$, key $k_h$ and $k_p$
Output: Table to be disclosed
Method:
1. $Emd\_Complete(T, n, k_h, k_p)$
2. $j = |\ grp(mas^{-1}(n-1))\ | * n$
3. For $i = |\ grp(mas^{-1}(n-1))\ |$ to $|\ grp(mas^{-1}(0))\ |$
4.     Let $W[j] = M[i][|\ seq(i)\ | -1]$
5.     $j = j + 1$
6.     For $k = 0$ to $|\ seq(i)\ | -2$
7.         Let $W[j] = M[i][k]$
8.         $j = j + 1$
9. Return $per(W, k_p)$
```

$|\ grp(1)\ | = 9$, $|\ grp(2)\ | = 8$, $|\ grp(3)\ | = 6$, $|\ grp(4)\ | = 3$, $|\ grp(5)\ | = |\ grp(6)\ | = 2$, and the remaining groups are all empty. The incomplete watermark sequences with reference should be:$(6, 0, 1, 2, 3, 4, 5), \ldots, (3, 0, 2), \ldots, (1, 0), (0)$

4.2.2 Verification

For verification, users will derive from k_h the key k_p. Firstly, users will use the key k_p to invert the permutation of tuples and obtain the watermarked table. Then from this watermarked table, user first attempt to obtain the master order function as previously mentioned. Then, users map each sequence index to a new value according to the master order function, and obtain the actual sequence indices. Users then construct expected sequence indices and compare them to the actual sequence indices in order to detect and localize attacked tuples.

Algorithm 4 describes the watermark verification algorithm for this first method. For complete sequences, the construction of expected sequence indices is the same as in the special case. However, for incomplete sequences, we must use the current table's information and the watermark embedding method to construct expected sequence indices. In

Line 3 and 5, users compute all tuples' sequence indices and group these indices, which are placed into M. For each group $grp(mas^{-1}(i))$ starting from $grp(mas^{-1}(n-1))$, if the group is not empty, users take i as the reference tuple's sequence index so the remaining tuples' indices in this sequence would be: $(0, 1, \ldots, i-1)$.

If we choose a sequence index j from group $grp(mas^{-1}(j))$, $grp(mas^{-1}(j))$'s size will be reduced. This construction method is thus similar to the embedding method. We first assume the incomplete sequences in V to be identical to those in the original tuples, and try to construct the incomplete watermark sequences. If table V is tampered, then the expected sequence indices we have constructed would be different from the values in the original watermarked table. By regarding the tuples' sequence indices in the current table L and the expected sequence indices I as two strings, we can apply the $LevenshteinDistance$ procedure to localize attacked tuples. In this algorithm, when we use Levenshtein distance to locates attacks' set, the worst-case complexity is $O(m^2)$, so the verification algorithm's time complexity is $O(m^2)$.

4.3 Algorithm 2

A key limitation of the first method lies in the fact that the construction of expected sequence indices critically depends on the length indicator of each incomplete sequence. Therefore, an attached length indicator may lead to a significant increase in the rate of false positives and negatives. This limitation is largely due to the varying sizes of incomplete sequences. Therefore, the key idea of our second method is to keep a fixed size n for every sequence in the watermarked table, except the last one (which may include less than n tuples).

In a complete watermark sequence $seq(i)$, each position $k \in [0, n-1]$ is filled by a tuple with corresponding sequence index k. In an incomplete watermark sequence, some

Algorithm 4 Algorithm Ver_Alg1

```
Input: Disclosed table $U$, module $n$, key $k_h$ and $k_p$
Output: Attached tuples
Method:
1.  $Ver\_Complete(U, n, k_h, k_p)$
2.  $j = q * n$ // $q$ is the complete sequence number
3.  For $i = j$ to $| V |$ // $V$ is the watermark table
4.      Let $L[i] = mas(sid(V[i]))$
5.      Let $M[gid(V[i])][mas(sid(V[i]))] = V[i]$
6.  For $i = n - 1$ to 0
7.      While $| grp(mas^{-1}(i)) | \neq 0$
8.          Let $I[j] = i$
9.          $| grp(mas^{-1}(i)) | = | grp(mas^{-1}(i)) | -1$
10.         $j = j + 1$
11.         For $k = 0$ to $i$
12.             Let $I[j] = k$
13.             $| grp(mas^{-1}(k)) | = | grp(mas^{-1}(k)) | -1$
14.             $j = j + 1$
15. Return $LevenshteinDistance(I, L)$
```

positions are empty, as we cannot find the tuples with corresponding sequence index and group index. Therefore, to rearrange tuples in incomplete sequences into sequences of size n, we have to *replace* a missing tuple with an existing tuple according to some publicly known rules, namely, *replacement rules.*

Definition 10 *A pseudo-complete sequence is a watermark sequence of size n that yield the same continuous sequences indices as complete sequences, by applying pre-defined replacement rules.*

4.3.1 Watermark Embedding

Definition 11 *Complementary positions: In a watermark sequence $seq(i)$ of size n, $M[i][a]$ and $M[i][b]$ are called complementary positions when $(a+b)$ mod $(n-1)=0$.*

We divide one complete watermark sequence into two parts. Let $mid = \lceil (n-1)/2 \rceil$. The sequence indices from 0 to mid are said to be in the left part, whereas the right part refers to $mid+1$ to $n-1$. Two complementary positions are thus always in different parts. Due to the master order function, in incomplete sequences, we have the property $\mid grp(mas^{-1}(0)) \mid \geq \mid grp(mas^{-1}(1)) \mid \geq \ldots \geq \mid grp(mas^{-1}(n-1)) \mid$. Therefore, the number of tuples in the left part is always no less than that in the right part.

The basic idea of our replacement rules is as follows. First, we use tuples in the left part to fill corresponding empty positions in the right part at the complementary position. Second, we use a pair of tuples to replace another pair, if the former is the next available pair of tuples. This second replacement rule is circular. For example, if the complementary position-based replacement finds the tuple from $grp(mas^{-1}(a))$ to replace position b, but $grp(mas^{-1}(a))$ is now empty, then we handle this situation in two steps. We first find a tuple from $grp(mas^{-1}(c))$ with the second method for replacement for position a, then we use a tuple from $grp(mas^{-1}(d))$ to replace position b (c and d are complementary positions where $c < d$).

Definition 12 *Define a replacement function* $rep(.) : M \longrightarrow M$ *such that* $rep(M[i][j]) = t$ *if either* $M[i][j]$ *is empty or* $mas(sid(t)) \neq j$ *and* t *is a tuple decided by the following replacement rules.*

1. **Complementary replacement:** *If* $M[i][a], M[i][b] \in seq(i)$, $M[i][b]$ *is empty or* $\mid grp(mas^{-1}(a)) \mid \neq 0$ *(*a *and* b *are complementary positions where* $a < b$*), then* t *is the next available tuple from* $grp(mas^{-1}(a))$*.*

2. **Shifting replacement:** *If* $M[i][a], M[i][c] \in seq(i)$, $M[i][a]$ *is empty (*$a \leq mid$*), then* t *is the next available tuple from the group* $grp(mas^{-1}(c))$ *where* c *is either from* $(a, mid]$ *or, if this is not possible, from* $[0, a)$*.*

Algorithm 5 describes the watermark embedding algorithm for this method. Line 1 calls the $Emd\_complete$ function and obtains $\mid grp(mas^{-1}(n-1)) \mid$ complete sequences. Line 2 to 6 construct pseudo-complete sequences. For each position i in an incomplete watermark sequence in M, if this position has a tuple, we place the tuple into W; otherwise, we use the function $per(.)$ to find another tuple to be placed into this position, and then place the tuple into W. Line 8 does the extra pass of permutation. The time complexity for this algorithm is also $O(m)$, depending on the micro-data table's size m.

For example, suppose $n = 10$, and in the incomplete sequence part, $\mid grp(0) \mid = 10$, $\mid grp(1) \mid = 9$, $\mid grp(2) \mid = 8$, $\mid grp(3) \mid = 6$, $\mid grp(4) \mid = 3$, $\mid grp(5) \mid = \mid grp(6) \mid = 2$. Figure 5 shows the result of pseudo-complete sequences.

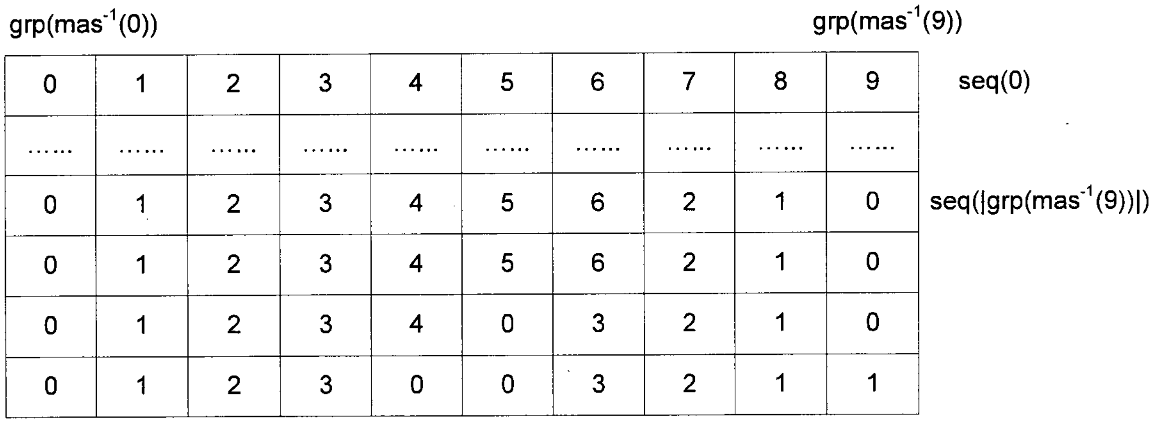
Proposition 1 *Algorithm 5 can always terminate, with the result containing either complete or pseudo-complete sequences except the last sequence.*

Proof: In the algorithm, if a position in the right half is empty, we will use tuples from the left half to fill this position. Therefore, the algorithm can continue as long as a tuple can

Algorithm 5 Algorithm *Emd_Alg2*

Input: Original table T, module n, key k_h and k_p
Output: Table to be disclosed
Method:

```
1. $Emd\_Complete(T, n, k_h, k_p)$
2. For $i = \mid grp(n-1) \mid *n$ to $m$
3.     If $M[\lfloor i/n \rfloor][i \bmod n] \neq NULL$
4.        Let $W[i] = M[\lfloor i/n \rfloor][i \bmod n]$
5.     Else
6.        Let $W[i] = rep(M[\lfloor i/n \rfloor][i \bmod n])$
7. Return $per(W, k_p)$
```



grp(mas^{-1}(0))									grp(mas^{-1}(9))	
0	1	2	3	4	5	6	7	8	9	seq(0)
……	……	……	……	……	……	……	……	……	……	
0	1	2	3	4	5	6	2	1	0	seq(|grp(mas^{-1}(9))|)
0	1	2	3	4	5	6	2	1	0	
0	1	2	3	4	0	3	2	1	0	
0	1	2	3	0	0	3	2	1	1	

Figure 5: An example of embedding watermarks with Algorithm 5

be found in the left part. As to the left part, since the shifting replacement rule is circular, the algorithm can continue until all tuples in the left part are exhausted. However, if the left half runs out of tuples before the algorithm processes all the tuples except those in the last sequence, then the algorithm will terminate without producing the desired result. Therefore, we need to prove that the left part will never run out of tuples before the right part does. We use mathematical induction on the index of incomplete sequences.

1. **Base case:** Unless the first incomplete sequence coincides with the last, by the definition of the master order function, we have that for any complementary positions a and b ($a < b$), $\mid grp(mas^{-1}(a)) \mid \geq \mid grp(mas^{-1}(b)) \mid$ must hold, so the result holds for the base case.

2. **Inductive case:** The inductive hypothesis is that after constructing n pseudo-complete sequences, if we can still have $\mid grp(mas^{-1}(a)) \mid \geq \mid grp(mas^{-1}(b)) \mid$, then it will remain true after constructing $n + 1$ pseudo-complete sequence.

   Consider the following cases.

   (a) If in the current sequence, no tuple in $grp(mas^{-1}(a))$ and $grp(mas^{-1}(b))$ will be chosen to replace others, then each group will just need one tuple, so $\mid grp(mas^{-1}(a)) \mid \geq \mid grp(mas^{-1}(b)) \mid$ is true.

   (b) If there is an complementary replacement in the two groups, then $\mid grp(mas^{-1}(b)) \mid = 0$ must be true; therefore, $\mid grp(mas^{-1}(a)) \mid \geq \mid grp(mas^{-1}(b)) \mid$ holds.

   (c) If there is an shift replacement, and if we use a tuple in $grp(mas^{-1}(a))$ to replace some other position, then we should also use tuple in $grp(mas^{-1}(b))$ to replace the complementary position, unless $grp(mas^{-1}(b))$ is empty. Therefore, group $grp(mas^{-1}(b))$ will never need less tuples than group $grp(mas^{-1}(a))$, and $\mid grp(mas^{-1}(a)) \mid \geq \mid grp(mas^{-1}(b)) \mid$ is thus true.

(d) Since adding both sides of the inequations in above cases 2 and 3 will not change the direction of the inequality, any combination of shift replacements and complementary replacements will satisfy $| grp(mas^{-1}(a)) | \geq | grp(mas^{-1}(b)) |$.

Therefore, after constructing $n+1$ pseudo-complete sequences, the left part will have more tuples than the right part.

This concludes the proof. □

4.3.2 Verification

The first step of verification procedure is the same as it in algorithm 1. The user keeps two keys k_h and k_p. Firstly, user uses the key k_p to recover tuples' order in watermarked table. From this received watermarked table, user can get the master order function. Then user maps each sequence index to a new value according to the master order function, and get the current table's sequence indices. Then user construct the expected sequence indices and compare them to detect and locate attacks. The difference between these two algorithms is the construction method for expected sequence indices.

Algorithm 6 describes the verification algorithm for this method. For complete sequences, the construction of expected sequence indices is the same as in the special case. Line 1 calls the $Ver\_Complete$ function to compute expected sequence indices and current sequence indices for complete watermark sequences. Line 3 and 5 compute all the tuples' sequence indices and group indices in the incomplete part, and place these tuples into M. Line 6 to Line 10 use the current table's information and watermark embedding method to construct expected sequence indices. For each position i in complete watermark sequence in M, if current position has a tuple, we compute this tuple's sequence index and place it into I. Otherwise, we will find another sequence index that satisfy the replacement rule and put it in I. After the construction, we regard the tuples' sequence indices in current table

L and the expected sequence indices I as two strings, and use the $LevenshteinDistance$ function to localize attacked tuples. In this algorithm, when we use Levenshtein distance to locates attacks' set, the worst-case complexity is $O(m^2)$, so the verification algorithm's time complexity is $O(m^2)$.

Algorithm 6 Algorithm Ver_Alg2

```
Input: The disclosed table $U$, module $n$, key $k_h$ and $k_p$
Output: Attached tuples
Method:
1.  $Ver\_Complete(U, n, k_h, k_p)$
2.  $j = q * n$ // $q$ is the complete sequence number
3.  For $i = j$ to $| V |$ // $V$ is the watermark table
4.      Let $L[i] = mas(sid(V[i]))$
5.      Let $M[gid(V[i])][mas(sid(V[i]))] = V[i]$
6.  For $i = j$ to $| V |$
7.      If $M[\lfloor i/n \rfloor][i \ mod \ n] \neq NULL$
8.        Let $I[i] = mas(sid(M[\lfloor i/n \rfloor][i \ mod \ n]))$
9.      Else
10.       Let $I[i] = mas(sid(rep(M[\lfloor i/n \rfloor][i \ mod \ n])))$
11. Return $LevenshteinDistance(I, L)$
```

4.4 Algorithm 3

For complete watermark sequences, we can construct expected sequence indices without current table's information. However, with algorithms 1 and 2, the expected sequence indices are deduced based on actual tuples. Therefore, the resulted expected sequence indices may be different from the actual ones, and the verification result will be negatively impacted. Therefore, for the next algorithm, we extract information, such as the length and number of sequences with the same length, about the incomplete watermark sequences and then embed such information separately into the table in a way similar to that of embedding the master order.

4.4.1 Embedding Auxiliary Information

We first discuss the method for embedding auxiliary information, such as the length of an incomplete sequences. The embedding method is similar to that of embedding the master order. Specifically, we can imagine a large array of sequences E each of which is a permutation over o integers $\{0, 1, \ldots, o-1\}$. We can then embed an integer j ($j \in [0, o!)$) by re-arranging the first o tuples in table W such that their sequence indices form the sequence $E[j]$.

Definition 13 *Denote by y, o, and r three integers. Define a function $emd(.)$ that embeds y into a table x of $o * r$ tuples. Each of the o tuples in x are arranged with the sequence indices ordered as $E[y]$, where E is a publicly known array, and repeated for r times.*

For example, suppose $E = (t_0, t_1, \ldots, t_{11})$ where $o = 3$, $r = 4$, $E[y] = (0, 2, 1)$. We have that $E(x, y, r) = (t_0, t_3, t_6, t_9, t_2, t_5, t_8, t_{11}, t_1, t_4, t_7, t_{10})$.

We shall denote the auxiliary information to be embedded, which is a set of integers, by k_r; for each integer $i \in k_r$, we map it to the order $E[i]$, and we then use the function $E(x, j, r)$ to reorder tuples in W based on $E[i]$.

4.4.2 Watermark Embedding

When we convert all tuples from T to the two dimensional array M, we can obtain $\mid grp(mas^{-1}(n-1)) \mid$ complete watermark sequences and place them directly into the watermark table W. For each of the remaining incomplete watermark sequence in M, the sequence indices are continuous and start from 0, so we only need to keep the size of each incomplete watermark sequence. Users can deduce each sequence's expected sequence indices. We define the auxiliary information k_r as two arrays, len storing the different sizes of incomplete watermark sequences and num storing the number of sequence with this size. We need to embed k_r by the method we discussed above.

Algorithm 7 describes the watermark embedding for k_r in table W. We choose each $o*r$ tuples as the table x. Line 1 embeds the length of $\mid k_r \mid$ by the function $emd(W[0, \ldots, o*r-1], \mid k_r \mid, r)$. Line 3 to 6 embed each value in array len and num. In this function, E, o and r are public information. The time complexity for this algorithm is also $O(m)$, depending on the micro-data table's size m.

Algorithm 7 Algorithm *Emd_Ref*

```
Input: Watermark table $W$, $k_r$
Output: Watermark table $W$ with $k_r$ embedded
Method:
1. Let $W[0, \ldots, o*r-1] = emd(W[0, \ldots, o*r-1], \mid k_r \mid, r)$
2. $j = o*r$
3. For $i = 0$ to $\mid k_r \mid$
4.     Let $W[j, \ldots, j+o*r-1] = emd(W[j, \ldots, j+o*r-1], len[i], r)$
5.     Let $W[j+o*r, \ldots, j+2o*r-1] = emd(W[j+o*r, \ldots, j+2o*r-1], num[i], r)$
6.     $j = j + 2o*r$
```

Algorithm 8 describes the watermark embedding algorithm for this method. Line 1 calls the *Emd_complete* function and gets $\mid grp(mas^{-1}(n-1) \mid$ complete sequences. Then for the incomplete part, Line 3 to 6 place them into W row by row, and collect the auxiliary information k_r. Then we call the *Emd_Ref* function to embed k_r into W. Line 8 permutes table M by key k_p, which can be disclosed.

4.4.3 Watermark Verification

The first step of verification procedure is the same as in the first method. The user derives the key k_p from k_h. Firstly, users use the key k_p to recover tuples' order in watermarked

Algorithm 8 Emd_Alg3

```
Input: Original table $T$, module $n$, key $k_h$ and $k_p$
Output: Table to be disclosed
Method:
1.  $Emd\_Complete(T, n, k_h, k_p)$
2.  $j = \mid grp(n-1) \mid *n$
3.  For $i = \mid grp(n-1) \mid$ to $\mid grp(0) \mid$
4.      For $k = 0$ to $\mid seq(i) \mid -1$
5.          Let $W[j] = M[i][k]$
6.          $j = j + 1$
7.  $Emd\_Ref(W)$
8.  Return $per(W, k_p)$
```

table. Then from the received watermarked table, users can get the master order function and k_r. Then users map each sequence index to a new value according to the master order function, and get the current table's sequence indices. From k_r, users can construct the expected sequence indices, then users compare them to detect and localize attacks.

Algorithm 7 describes the method for user to obtain k_r. Line 1 gets the length of k_r. Then for each $o * r$ tuples, users use the emd^{-1} function to get len and num. As some tuples in this part may be attacked, users use the majority voting scheme to decide the most plausible result. Then users compute this part's expected sequence indices from k_r place them in I.

Algorithm 10 describes the watermark verification algorithm for this method. In complete sequences, the construction method for expected sequence indices is the same as in the special case. Line 1 calls the $Ver\_Complete$ function and gets expected sequence indices and current sequence indices for complete watermark sequences. Line 2 calls the Ver_Ref function and gets k_r from table V. Then users can compute expected sequence indices from k_r.

For the incomplete part, we do not use the current table's information to construct

Algorithm 9 Algorithm Ver_Ref

```
Input: Watermarked table $V$
Output: Auxiliary information $k_r$
Method:
1.  $| k_r |= emd^{-1}(W[0,\ldots,o*r-1])$
2.  $j = o*r$
3.  For $i = 0$ to $| k_r |$
4.      Let len[i] = $emd^{-1}(W[j,\ldots,j+o*r-1])$
5.      Let num[i] = $emd^{-1}(W[j+q*r,\ldots,j+2o*r-1])$
6.      $j = j+2o*r$
7.  Let $I[0,\ldots,o*r-1] = sid(emd(W[0,\ldots,o*r-1], | k_r |, r))$
8.  $k = o*r$
9.  For $i = 0$ to $| k_r |$
10.     Let $I[k,\ldots,k+o*r-1] = sid(emd(W[k,\ldots,k+o*r-1], len[i], r))$
11.     Let $I[k+o*r,\ldots,k+2o*r-1] = sid(emd(W[k+o*r,\ldots,k+2o*r-1]$
        $, num[i], r))$
12.     $k = k+2o*r$
```

expected sequence indices, but use k_r instead. Each value $len[i]$ indicates the sequence size to be $len[i]$, and there are $num[i]$ such sequences. The sequence indices for a sequence with size $len[i]$ are: $(0,\ldots,len[i]-1)$. so we can compute all the expected sequence indices. After construction, we consider the tuples' sequence indices in the current table and the expected sequence indices I as two strings, and use *LevenshteinDistance* function to localize attacks. In this algorithm, when we use Levenshtein distance to locates attacks' set, the worst-case complexity is $O(m^2)$, so the verification algorithm's time complexity is $O(m^2)$.

Algorithm 10 Ver_Alg3

```
Input: Disclosed table $U$, module $n$, key $k_h$ and $k_p$
Output: Attached tuples
Method:
1.  $Ver\_Complete(U, n, k_h, k_p)$
2.  Ver_Ref(V) // $V$ is the watermark table
3.  $j = p * n$ // $p$ is the complete sequence number
4.  For $i = j$ to $| V |$
5.      Let $L[i] = mas(sid(V[i]))$
6.  For $i = 0$ to $| k_r |$
7.      For $k = 0$ to $num[i]$
8.          For $l = 0$ to $len[i]$
9.              Let $I[j] = l$
10.             $j = j + 1$
11. Return $LevenshteinDistance(I, L)$
```

4.5 Handling Updates

In this section, we explain how to insert, delete or modify a tuple by the owner or authorized user. When a fragile watermark already exists in the table, it may need to be changed when the current table is updated. If we simply rebuild all watermarks, then if the owner or authorized user need to frequently update the table, then the cost will be prohibitive. Therefore, we propose algorithms to address this issue. As modifications can be considered as pairs of insertions and deletions, we only discuss insertion and deletion.

4.5.1 Insertion

In the special case, which means all the tuples are in complete sequences, users only need to be add new tuples to the end of the table. For the general case, consider the three different algorithms presented above.

1. For the first method, suppose a tuple t is inserted. We firstly compute $mas(sid(t)) = x$ and then search from the beginning of incomplete sequences to find the position

where the length indicator tuple's sequence index is equal to $x - 1$. Then we insert t into this sequence as the new length indicator and re-arrange the remaining tuples in this sequence.

For example, suppose the current incomplete watermark sequences' sequence indices are: $(5, 0, 1, 2, 3, 4)$, $(4, 0, 1, 2, 3)$, $(3, 0, 1, 2)$, $(2, 0, 1)$. Suppose a user wants to insert t $(mas(sid(t)) = 4)$. So we search and find a length indicator tuple with sequence index 3. Then we insert t into this sequence and re-arrange the tuples as: $(5, 0, 1, 2, 3, 4)$, $(4, 0, 1, 2, 3)$, $(4, 0, 1, 2, 3)$, $(2, 0, 1)$.

2. For the second method, suppose a tuple t_a $(mas(sid(t_a)) = x)$ is inserted. If we put t_a at the end of the table, it may cause a conflict with the current replacement rules since the rules may assume this sequence index will not appear at this position. We should put the tuple into the first sequence where $grp(mas^{-1}(x))$ is empty. Suppose in this sequence, we put t_b at position x. Then we put t_a at this position, and t_b should then be put into the first position where $grp(sid(t_b))$ runs out of tuples.

   For example, suppose the current incomplete watermark sequence indices are: $(0, 1, 2, 3, 4, 5)$, $(0, 1, 2, 3, 4, 0)$, $(0, 1, 2, 3, 1, 2)$. A user wants to insert t_a $(mas(sid(t_a)) = 5)$. So we search the sequences and find the first position where $grp(mas^{-1}(5))$ is empty, and put t_a there. For the tuple with sequence index 0, which is put into position 5, we place it at the position where $grp(mas^{-1}(0))$ is empty. So the result will be $(0, 1, 2, 3, 4, 5)$, $(0, 1, 2, 3, 4, 5)$, $(0, 1, 2, 3, 1, 0)$, (2).

   In some special cases, the insertion will destroy the current master order. That is, for two groups, $| grp(mas^{-1}(x)) | \geq | grp(mas^{-1}(y)) |$ is true and the insertion changes this inequation. In this case, we will have to change the master order function and re-arrange all the tuples in the table.

3. For the third method, the insertion method is same as with algorithm 1, suppose

a tuple t ($mas(sid(t)) = x$) is inserted. We search from the beginning of incomplete watermark sequences and find the position where we run out of tuples in $grp(mas^{-1}(x))$. Then we insert t into this sequence and modify the k_r.

For example, the current incomplete part's sequence indices are $(0,1,2,3,4,5)$, $(0,1,2,3,4)$, $(0,1,2,3)$, $(0,1,2)$, $k_r = (6,1), (5,1), (4,1), (3,1)$. A user wants to insert t ($mas(sid(t)) = 4$), so we search and find the first position where $grp(mas^{-1}(4))$ runs out of tuples. Then we insert t into this sequence and change k_r. So the result is $(0,1,2,3,4,5)$, $(0,1,2,3,4)$, $(0,1,2,3,4)$, $(0,1,2)$, and $k_r = (6,1)\ (5,2)\ (3,1)$.

4.5.2 Deletion

In the special case, if a user wants to delete a tuple from a complete watermark sequence, this sequence will be destroyed. Therefore, the user must find a tuple with the same sequence index from the last sequence, and insert it into the position where the tuple is deleted. In the general case, we use the following methods to delete a tuple by considering the three different algorithms separately.

1. For method 1, if a tuple t_a ($mas(sid(t_a)) = x$) is deleted, we can find the last tuple t_b in group $grp(mas^{-1}(x))$ and put it into the deleted position, and then re-arrange the sequence in which t_b is included.

   For example, the current incomplete watermark sequences' sequence indices are: $(5,0,1,2,3,4)$, $(4,0,1,2,3)$, $(3,0,1,2)$, $(2,0,1)$. Suppose a user wants to delete t_a ($mas(sid(t_a)) = 4$) and t_a is in the first sequence. So we choose the last tuple t_b with sequence index 4 and put it into the first sequence, and then re-arrange the sequence where t_b is located. So the result is $(5,0,1,2,3,4)$, $(3,0,1,2)$, $(3,0,1,2)$, $(2,0,1)$.

2. For method 2, if a tuple t_a ($mas(sid(t_a)) = x$) is deleted, we can find the last tuple t_b with sequence index x and put it in t_a's position. We then find another tuple t_c

$(mas(sid(t_c)) = y)$ to replace t_b.

For example, the current incomplete watermark sequences' indices are $(0, 1, 2, 3, 4, 0)$, $(0, 1, 2, 3, 4, 0)$, $(0, 1, 2, 3, 1, 0)$, (0). Assume a user wants to delete t_a $(mas(sid(t_a)) = 4)$ and t_a is located in the first sequence. So we find another tuple with sequence index 4 from second sequence to replace it. Then in the second sequence, we run out of 4, so we have to use a tuple with sequence index 1 from the third sequence to replace it; we then run out of 1, so we need to use a tuple with sequence index 0 to replace it. Eventually, the result is $(0, 1, 2, 3, 4, 0)$, $(0, 1, 2, 3, 1, 0)$, $(0, 1, 2, 3, 0, 0)$.

3. For method 3, the deletion method is also the same as that for method 1. Suppose a tuple t_a $(mas(sid(t_a)) = x)$ is deleted. We find the last tuple with sequence index x and put it into the deleted tuples' position, and then we modify k_r and re-embed it.

4.6 Multi-Level Watermarking

In this section, we introduce a multi-level extension to increase the detecting probability. The first level of watermark embedding is already introduced above. For the second level, we use the following interleaved method.

In the basic scheme, we use the HMAC function with a key k_h to compute the keyed hash value for each tuple, which is then modulo with an integer n. After the re-ordering at first level, all tuples are divided into groups with the same sequence indices. However, among tuples inside each group (not sequence), there is essentially no order. Therefore, the second level of watermarking works inside these groups. Specifically, for all tuples inside each group, we use a new sequence position function $sid(t, k_{h'})$ to re-hash them by $k_{h'}$ and modulo the result with an integer n'. Then we reorder within each group according to the results. For the incomplete watermark sequence, the embedding method will be the same as for the first level.

Figure 6 shows an example for the interleaved algorithm. $t_1, t_2, t_3, \ldots, t_{24}$ denote tuples in a micro-data table. We use n=4 for the first level and n'=3 for the second level. $t_i(x_1, x_2)$ means a tuple t_i with an index x_1 for the first level and x_2 for the second level.

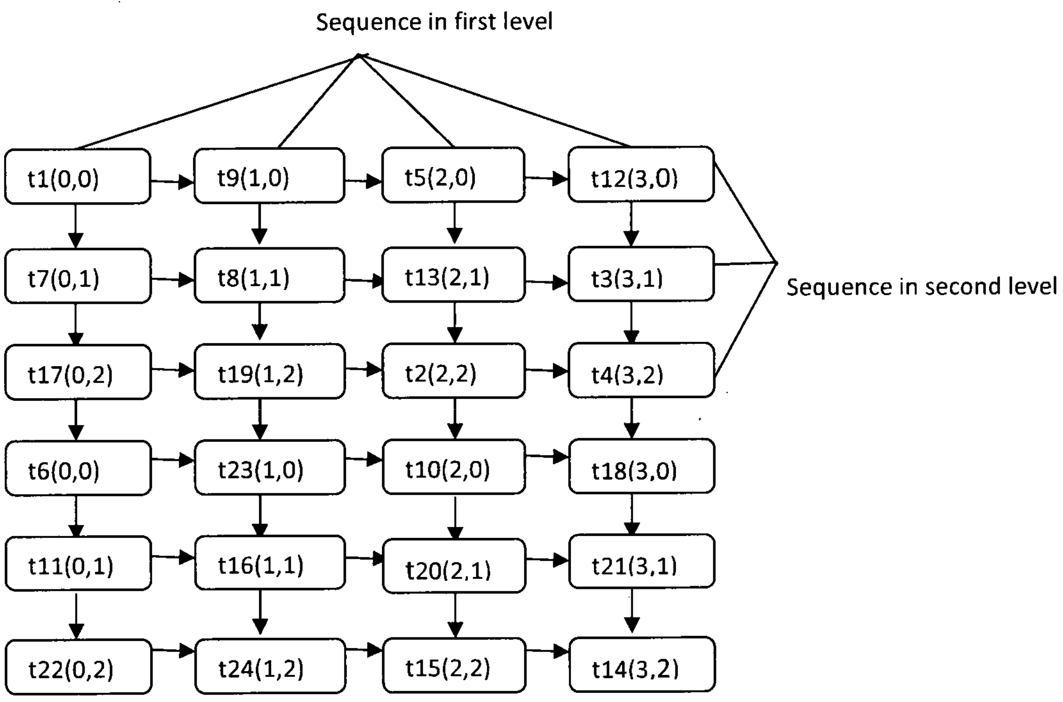


Figure 6: example–Interleaved

The two-level watermarking scheme will increase the detection probability since an attacked tuple must happen to have the same sequence index for both levels at the same time, which has the probability $1 - 1/n_1 \cdot n_2$. The detection probability for modifying l tuples is thus $1 - (1/n_1 \cdot n_2)^l$.

In theory, we can certainly repeat the same methodology to have the three and more levels of watermark embedding. For example, the third level will be to reorder those tuples with the same sequence indices for both the first and second level. However, since the embedding will be over a smaller and smaller number of tuples, it would become increasingly difficult to have complete sequences, and therefore the effectiveness will also decrease. In practice, the optimal number of levels will depend on the size of micro-data tables. We shall investigate the effectiveness of two-level watermark embedding in the next section through experiments.

Chapter 5

Experiments

We have implemented and tested the performance of our proposed techniques on machines equipped with Intel Pentium M 1.80GHz processor, 1024MB RAM, and Windows XP operating system. The main objective of the experiments is to compare the performance of different embedding schemes, to find the optimal parameters such as the modulus, and to evaluate the false positives and false negatives. We use the popular Adult Data Set taken from the UCI data repository [7].

5.1 How to Choose Modulus n

When we choose the modulus n, there are several aspects that need to be considered: The detection probability, the number of complete sequences, and the tuples that are used for storing the master order and auxiliary information. A big n leads to better detection probability but reduces the complete watermark sequences's number and also requires more tuples to keep the master order.

Figure 7 shows the percentage of tuples of complete watermark sequences in different module n. Here we obtain our data from the Adult dataset of UCI with $15,000$ tuples, and we vary n from 10 to $1,000$. For each n, we try different value of key k_h to get the

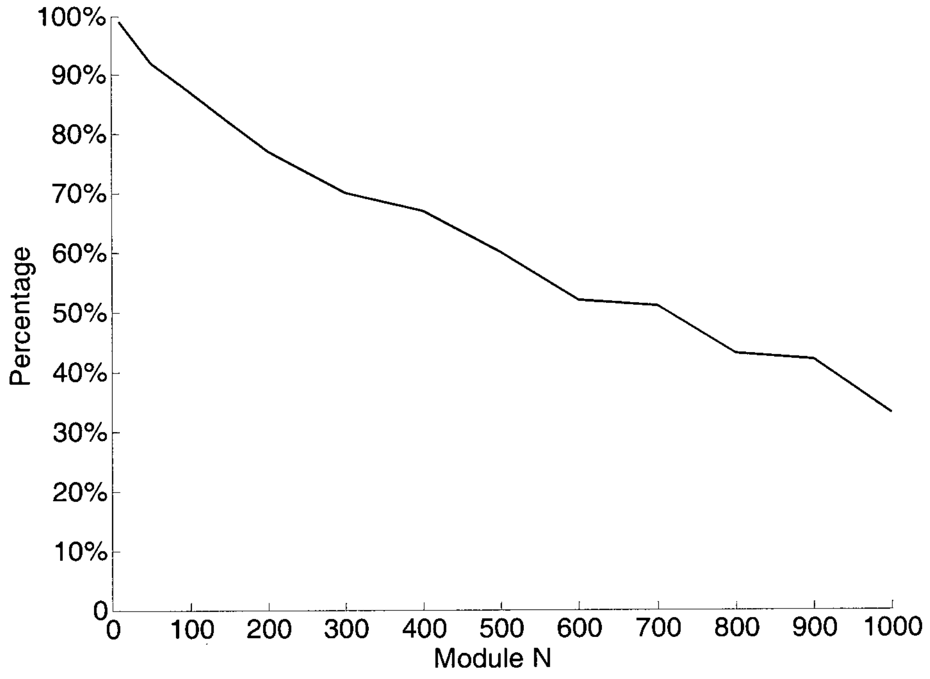


Figure 7: Percentage of Complete Sequences

maximum number of complete watermark sequences. From this graph, we can see when $n = 1000$, there are only 30% of tuples in complete watermark sequences, which is about 5 sequences.

In our algorithm, we need complete watermark sequences to embed master order and auxiliary information for method 3. Suppose we repeat each embedded integer 5 times, then at least we need more than 5 complete watermark sequences to embed all the information. Therefore, we should have n no more than 700. Besides, when n is small, the detection probability is negatively impacted. Therefore, in the following experiments, we choose n in the range $[50, 700]$.

5.2 Attacks in Complete Sequences

In this section, we study the localization capability in complete watermark sequences. We consider two aspects: The false positives in the number of tuples and the false negatives. We still use the same micro-data table with $15,000$ tuples, with varying n. For each n, we only choose the complete watermark sequences. Suppose the number of attacked tuples is $l = 50$. We consider the following attacks: Modification, deletion, insertion, and the combination of these.

Figure 8, 9 depict the false positive and false negative results for complete watermark sequences. The x-axis is the modulus n, and the y-axis the false positive and false negative results.

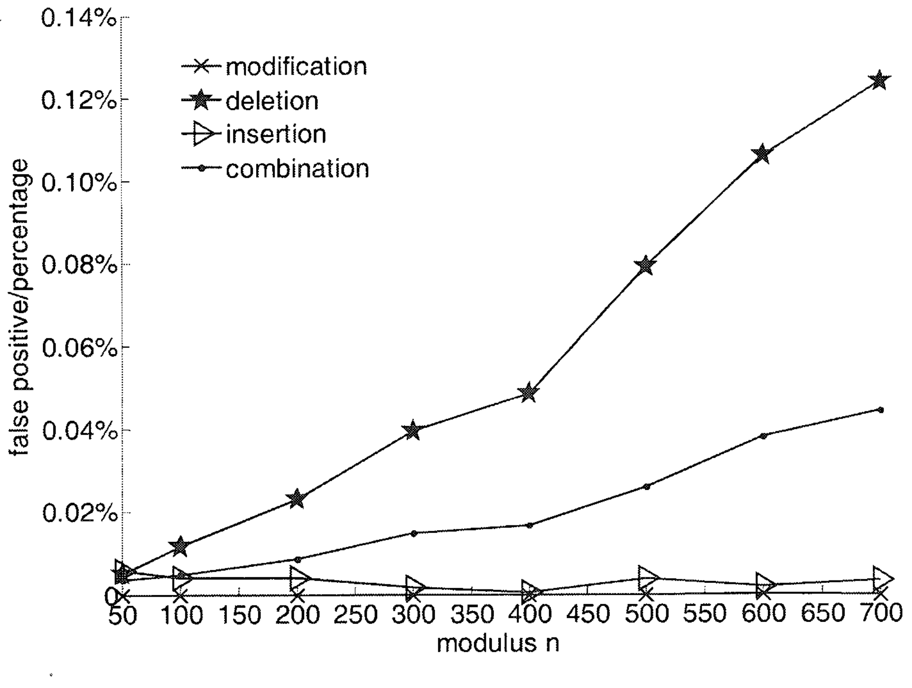


Figure 8: False Positives for Complete Sequences

From the results, we can see the false positives and false negatives for modification and insertion are not significant. For combination attacks and deletion, the false positive

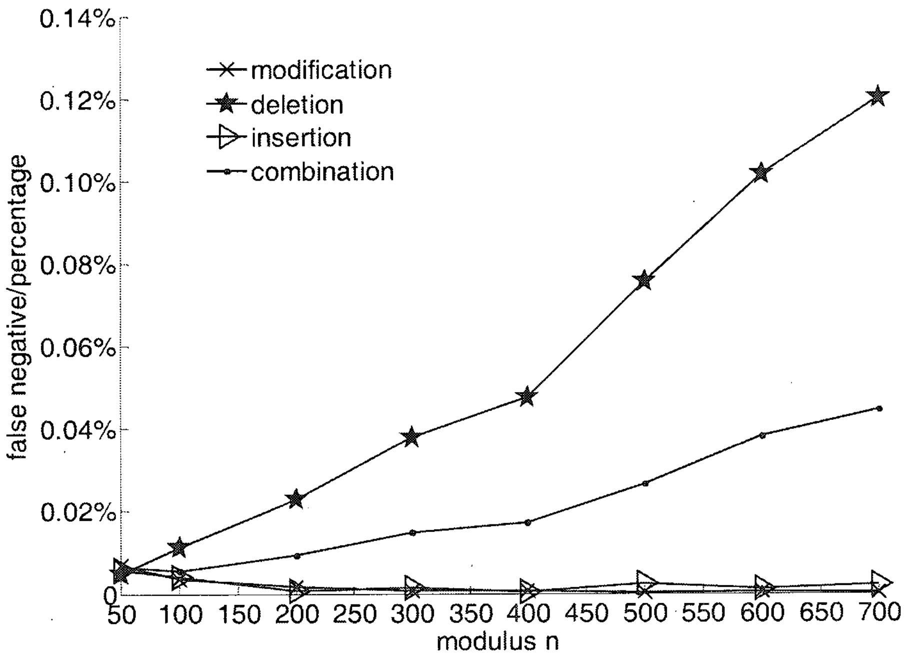


Figure 9: False Negatives for Complete Sequences

and false negative results are not as good (especially for deletion). From the algorithms, we can see that the localization for combination attacks will be less effective than that for known-type attacks since less information is available. So it is not surprising to see that combination attack's results are indeed worse than that of single known-type attack.

In our experiment, we repeat the master order for 5 times. So in the first 5 complete watermark sequences, each 5 tuples will have the same sequence index. This master order embedding method will weaken the localization capability, especially for deletion attack. The reason is the following. In the first 5 complete watermark sequences, when a tuple is deleted, we can detect and localize the deletion to be inside the five tuples, but we cannot exactly pinpoint which one is really deleted.

Moreover, in this experiment, we only choose complete watermark sequences from the micro-data table, so the first 5 complete watermark sequences may comprise a high percentage among all sequences. Therefore, the deletion attack's localization results are worse than others. However, since our purpose for localization is to eventually obtain the intact tuples from an attacked micro-data table so the tuples can still be useful, we consider the localization of deleted tuples to be of less importance.

5.3 Attacks on Different Algorithms

In this part, we compare the localization capability for different algorithms. We still use the micro-data table with $15,000$ tuples, and we choose the number of attacked tuples $l = 50$ and the modulus n in range $[50, 700]$.

Figure 10, 11, 12, 13, 14, 15, 16, 17 show false positive and false negative results for modification, deletion, insertion, and the combination of these attacks. In each chart, we compare the results between the three different methods.

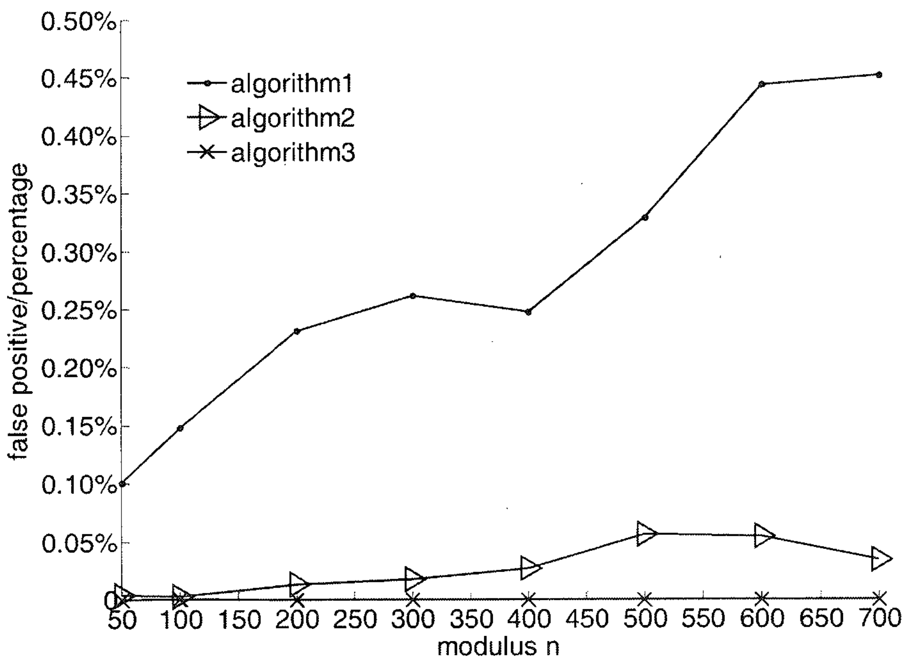


Figure 10: False Positive for Modification

From these results, we can conclude that, method 1's localization capability is much

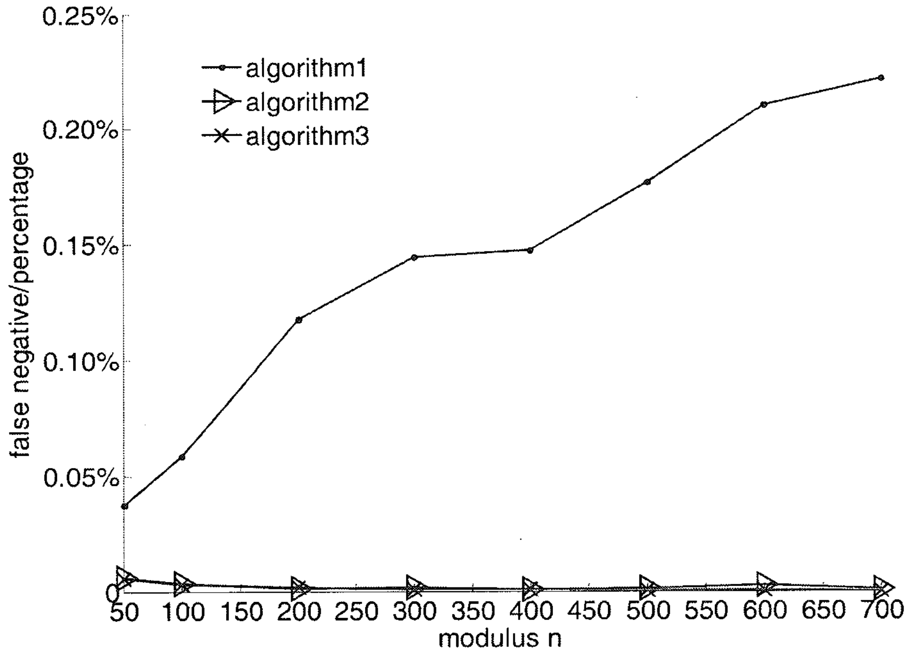


Figure 11: False Negative for Modification

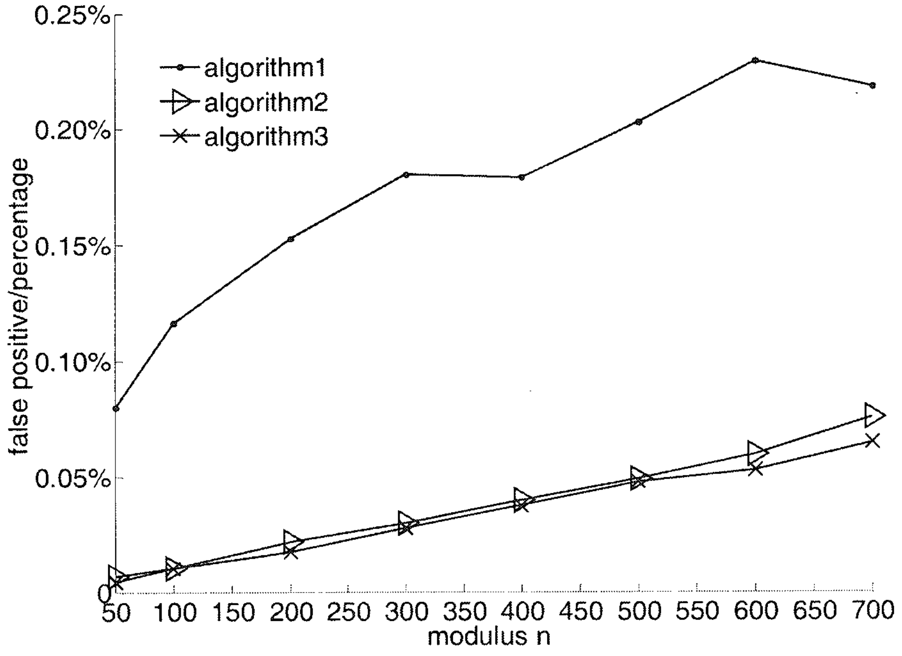


Figure 12: False Positive for Deletion

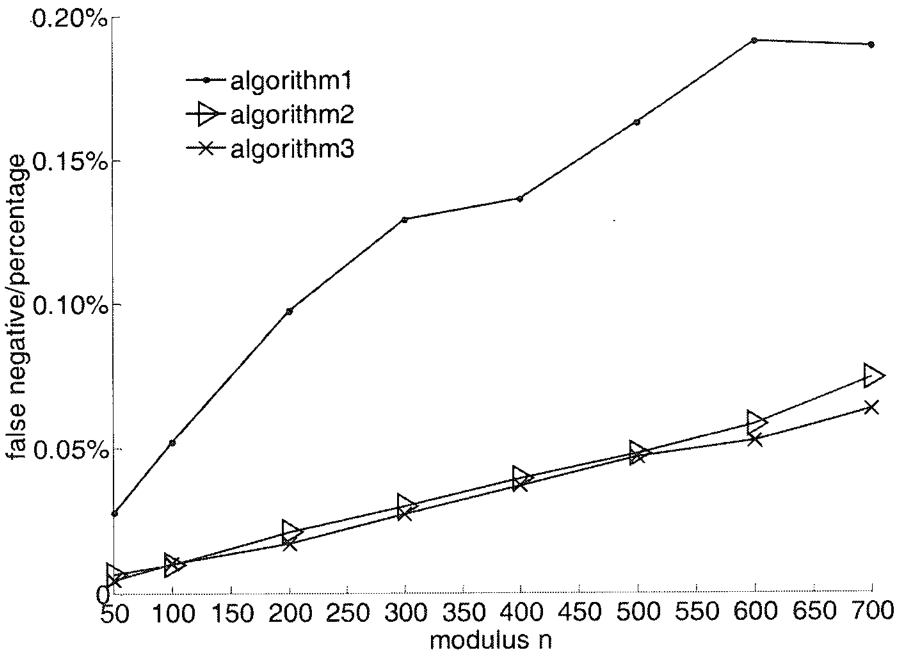


Figure 13: False Negative for Deletion

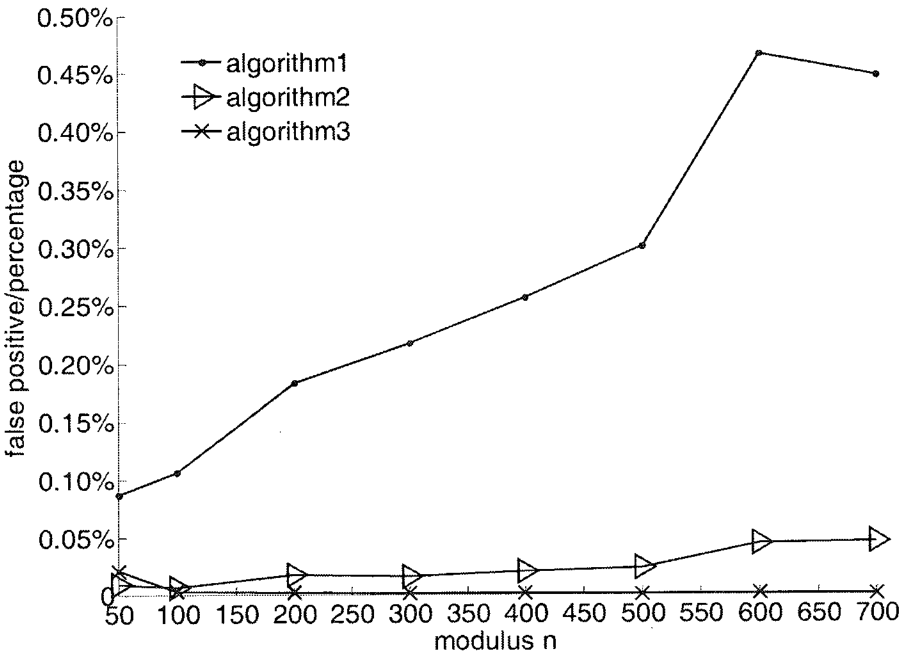


Figure 14: False Positive for Insertion

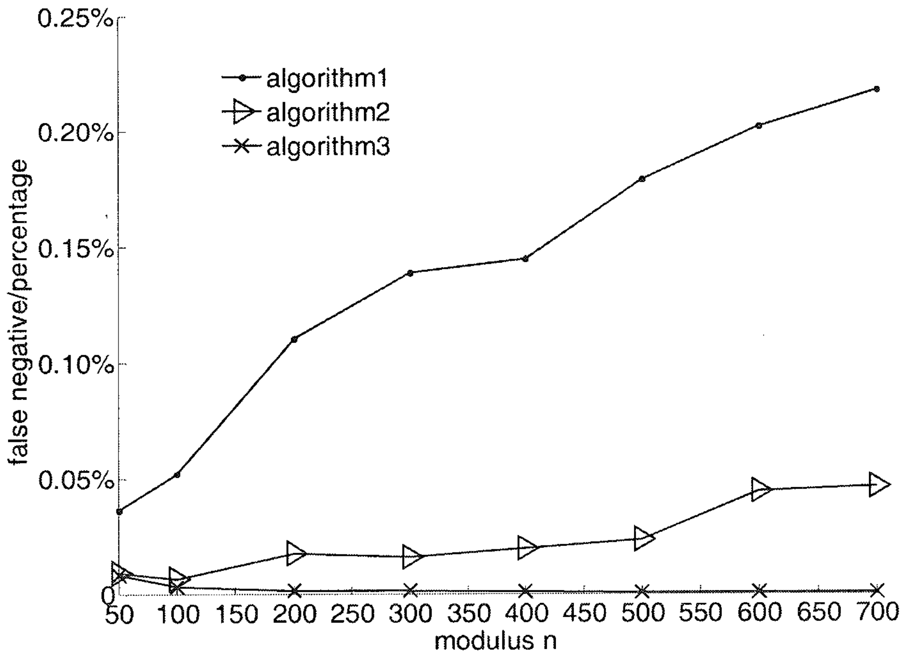


Figure 15: False Negative for Insertion

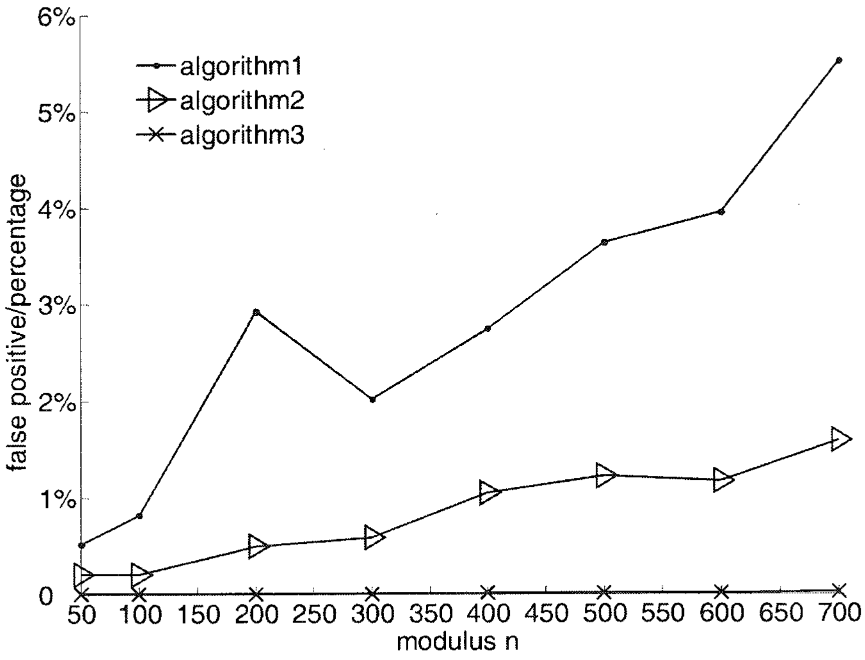


Figure 16: False Positive for Combination

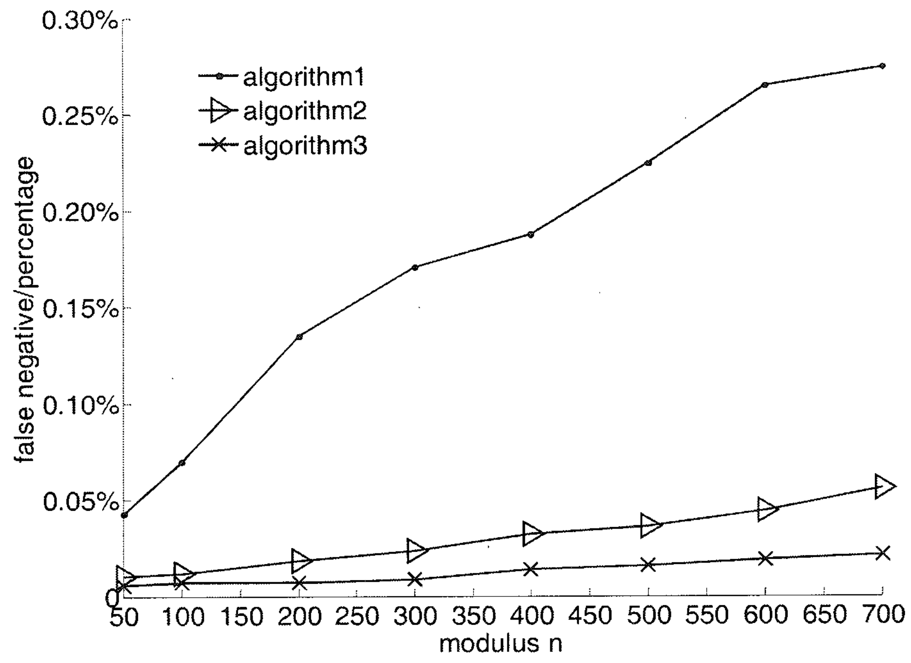


Figure 17: False Negative for Combination

worse than that of other methods. Method 3 overall yields the best results. The results for method 3 are close to the results in complete watermark sequences. In method 3, the main factor that affects the localization capability is still the tuples that are used to embed the master order.

Comparing the results in method 3 and those of complete watermark sequences, when we both attack 50 tuples, in complete watermark sequences, we can see that there are more attacks located in the first 5 sequences. Therefore, for deletion attack, method 3's false negative result is even better than that in complete watermark sequences.

Also, for method 1 and method 2, a big modulus n will affect the expected sequence indices, and the localization results will also be affected. Therefore, when using these two algorithms, we should not choose a large n.

Now we compare different attacks on algorithm 3. For modification attack and insertion attack, when n increases, the detection probability increases as well, so the false negative result will be better. However, when n increase, more tuples are needed to embed master order, so the false negative results for deletion attack and combination attack will get worse. Therefore, n cannot be either too small or too big. Through these experiments results, we choose $n = 400$ for method 3.

5.4 Localization Capability

From previous experiments, we find that method 3 has the best localization capability. So we test the boundary of this method's localization capability as the maximum number of attacks that this method can handle. We choose the modulus $n = 400$, and try different number of attacks l. We only test combination attack.

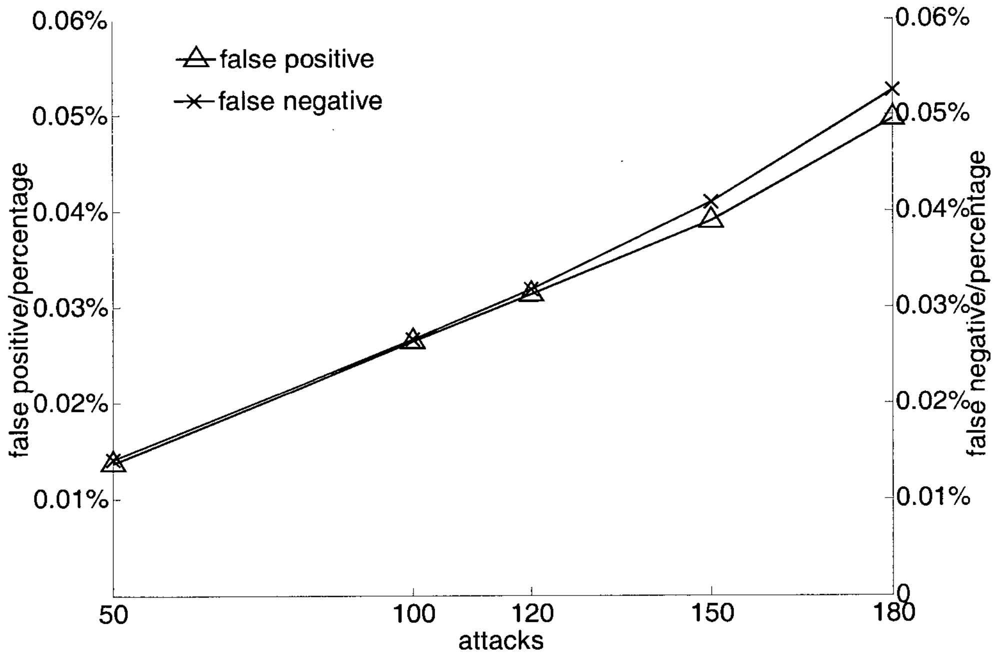


Figure 18: Attacks for Method 3

Figure 18 shows the localization results for different l in combination attack with method 3. The x-axis is the number of attacked tuples. The two y-axis are false positives and false negatives. We choose l from 50 to 180. When $l > 180$, we cannot get right master order and auxiliary information so we cannot carry out localization at all. When $l \leq 180$, method 3 can get good localization results.

5.5 Multi Level

Figure 19 describes the percentage of complete sequences with the two-level embedding scheme. The x-axis is the first level's modulus n_1. When we have two levels, in order to get more complete watermark sequences, the two levels' modulus cannot be too big. So in the first level, we choose n_1 in range $[0, 500]$. Besides, the second level's modulus n_2 must be less than n_1 so we choose $n_2 = 10$. From this figure we can see that when $n_1 > 300$, we essentially cannot get any complete sequence at the second level.

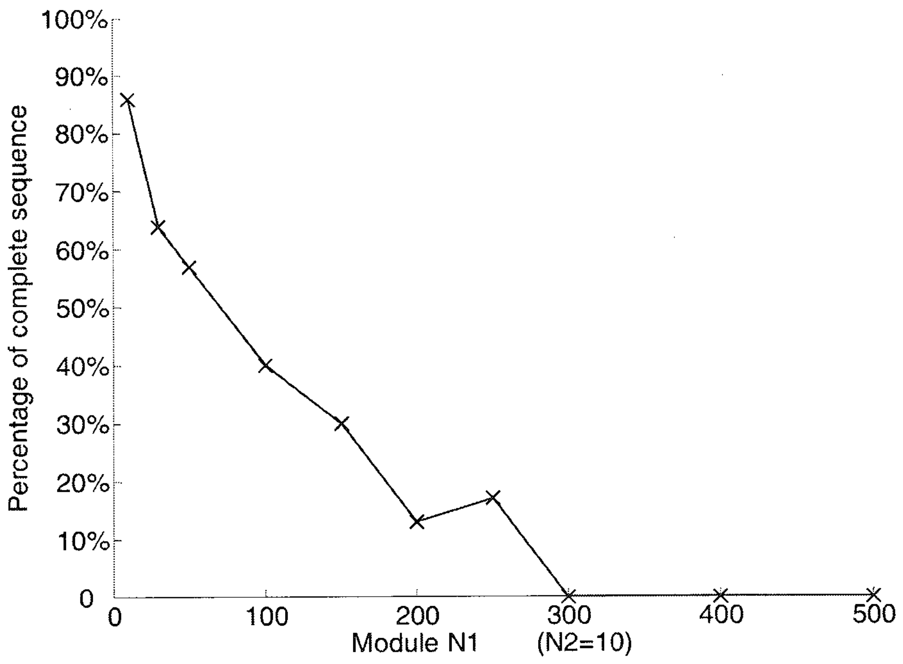


Figure 19: Complete Percentage for Second Level

The previous experiment with complete watermark sequences shows that the false positive results are good. Besides, our localization objective means the false negative results are relatively more important. So for the two level scheme, we shall only compare the false negative results between different methods.

Figure 20, 21, 22, 23 compare the false negative results for single level and two-level scheme. We still use the micro-data table with $15,000$ tuples, and choose the second level's modulus $n_2 = 10$, and the number of attacks $l = 50$. The x-axis is the first level's modulus n_1, chosen from $[0, 250]$. The y-axis is the false negative results.

We compare the three single known-type attacks and the combination of these attacks. We only consider the special case that, in both levels, all tuples are in complete watermark sequences so we do not embed the master order. The figures show, for all types of attacks, the two level's model can get less false negative tuples than single level scheme does.

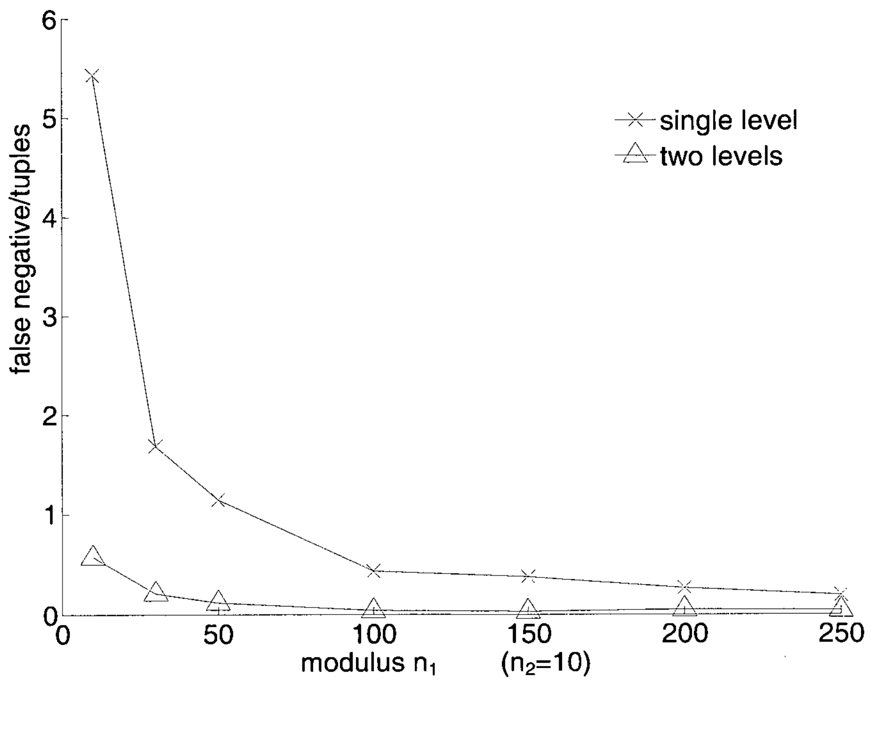


Figure 20: Modification Attack for Single level & Two Level

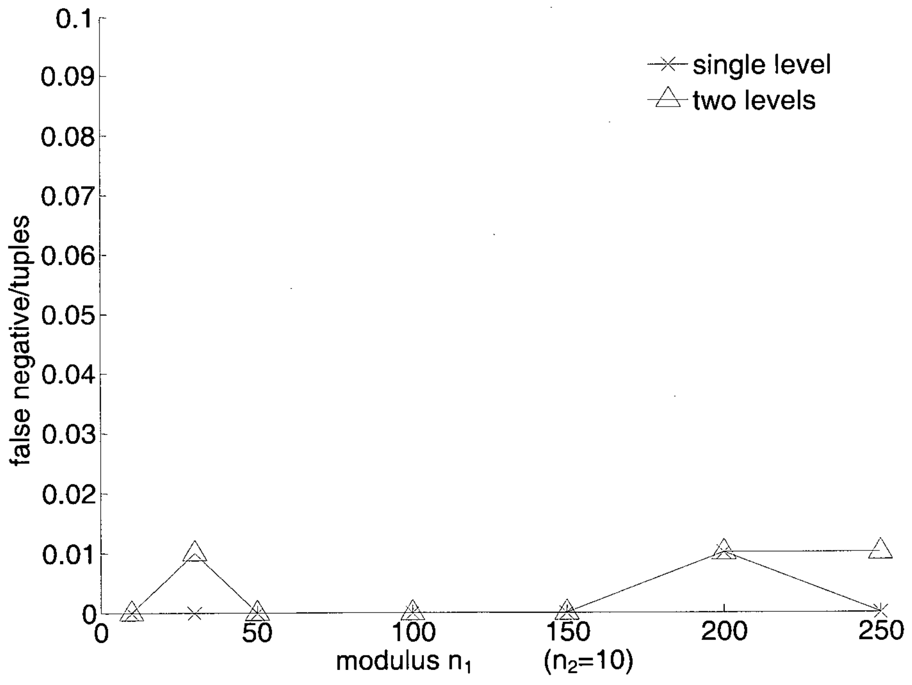


Figure 21: Deletion Attack for Single level & Two Level

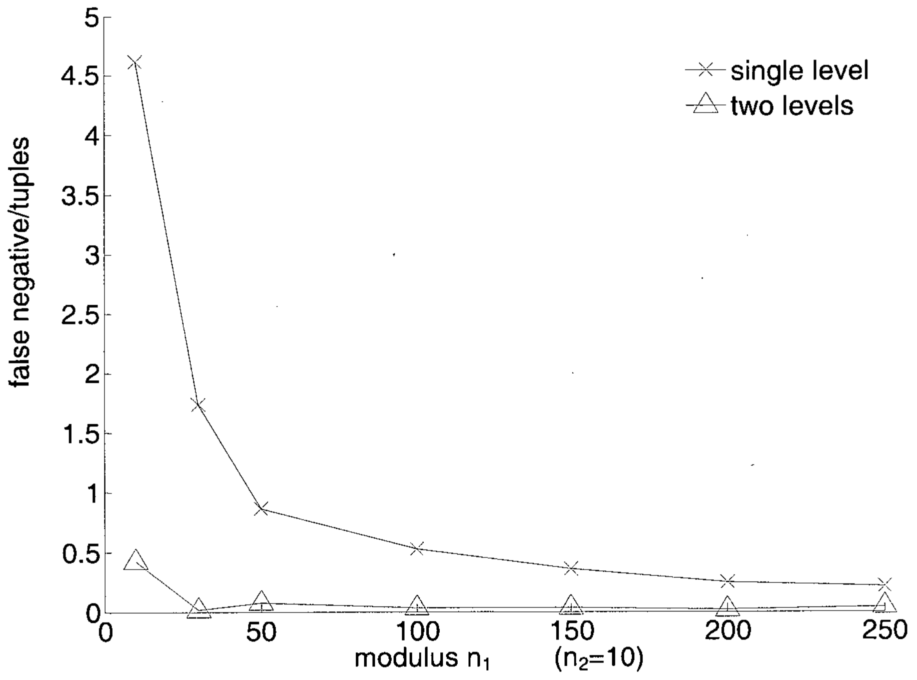


Figure 22: Insertion Attack for Single level & Two Level

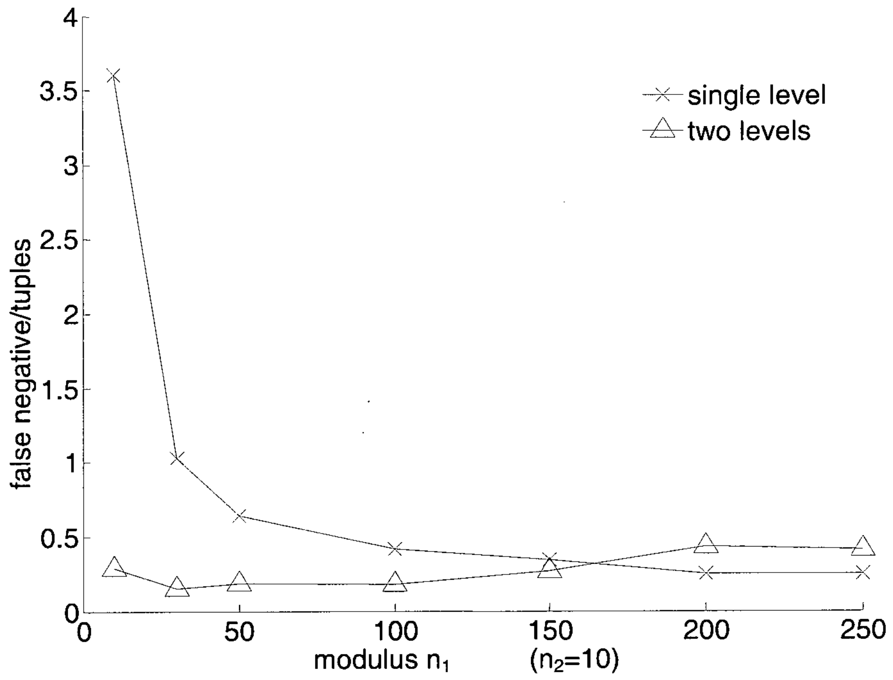


Figure 23: Combination Attack for Single level & Two Level

Chapter 6

Conclusion

In this thesis, we have discussed the issue of tamper detection and localization in disclosed micro-data tables. We introduced a series of methods to ensure the integrity of micro-data tables. In addition to the basic scheme for complete sequences, we designed three different algorithms to embed watermarks for incomplete sequences, based on the order of tuples. For each of these algorithms, we give verification algorithms to detect and localize attacked tuples. In this thesis, we have considered different types of attacks and combinations of attacks, all of which can be addressed by our proposed watermarking algorithms.

Compared to other signature and Merkel Hash Tree-based methods, our methods will not require the sever to keep any additional information, such as signatures, so the storage requirement is significantly reduced. Besides, our watermarking methods are based on hash functions, which is more efficient than public-key cryptographic methods. Compared to other existing watermarking-based integrity methods, our proposed scheme hashes each tuple independently, so attacked tuples can potentially be localized to the granularity of a single tuple.

Our experiments compared the proposed algorithms. The results show that our method can provide an acceptable rate of false positives and false negatives. Our future work include the study of other applications of the order-based watermarking techniques. Also, we would like to study more efficient methods for handling updates when the multi-level watermarking scheme is in place. Finally, we plan to investigate the performance of the proposed methods on different sizes of databases to find an optimal way to choose the modulus parameter.

Bibliography

[1] A.Dobra and S.E.Feinberg. Bounding entries in multi-way contingency tables given a set of marginal totals. In *Foundations of Statistical Inference: Proceedings of the Shoresh Conference 2000*. Springer Verlag, 2003.

[2] G. Aggarwal, T. Feder, K. Kenthapadi, R. Motwani, R. Panigrahy, D. Thomas, and A. Zhu. Anonymizing tables. In *ICDT'05*, pages 246–258, 2005.

[3] G. Aggarwal, T. Feder, K. Kenthapadi, R. Motwani, R. Panigrahy, D. Thomas, and A. Zhu. Approximation algorithms for k-anonymity. *Journal of Privacy Technology*, November 2005.

[4] Rakesh Agrawal and Jerry Kiernan. Watermarking felational databases. In *VLDB '02: Proceedings of the 28th International Conference on Very Large Data Bases*, pages 155–166. VLDB Endowment, 2002.

[5] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order preserving encryption for numeric data. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 563–574, New York, NY, USA, 2004. ACM.

[6] A.Slavkovic and S.E.Feinberg. Bounds for cell entries in two-way tables given conditional relative frequencies. *Privacy in Statistical Databases*, 2004.

[7] A. Asuncion and D.J. Newman. UCI machine learning repository, 2007.

[8] Minghua Chen, Yun He, and Reginald L. Lagendijk. A fragile watermark error detection scheme for wireless video communications. *IEEE Transactions on Multimedia*, 7(2):201–211, 2005.

[9] Weiwei Cheng, HweeHwa Pang, and Kian-Lee Tan. Authenticating multi-dimensional query results in data publishing. *Data and Applications Security*.

[10] B. Chor, O. Goldreich, and E. Kushilevitz. Private information retrieval. *Journal of ACM*, 45:965–981.

[11] Ingemar J. Cox, Matt L. Miller, and Jeffrey A. Bloom. Watermarking applications and their properties. In *International Conference on Information Technology: Coding and Computing*, pages 6–10, 2000.

[12] Ernesto Damiani, S. De Capitani Vimercati, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Balancing confidentiality and efficiency in untrusted relational dbmss. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, pages 93–102. ACM, 2003.

[13] Diego F. de Carvalho, Rafael Chies, André P. Freire, Luciana A. F. Martimiano, and Rudinei Goularte. Video steganography for confidential documents: Integrity, privacy and version control. In *SIGDOC '08: Proceedings of the 26th Annual ACM International Conference on Design of Communication*, pages 199–206. ACM, 2008.

[14] A. Deutsch. Privacy in database publishing: a bayesian perspective. In Michael Gertz and Sushil Jajodia, editors, *Handbook of Database Security: Applications and Trends*, pages 464–490. Springer, 2007.

[15] A. Deutsch and Y. Papakonstantinou. Privacy in database publishing. In *ICDT*, pages 230–245, 2005.

[16] Prem Devanbu, Michael Gertz, Chip Martel, and Stuart G. Stubblebine. Authentic third-party data publication. In *14th IFIP 11.3 Conference on Database Security*, 1999.

[17] D.P.Dobkin, A.K.Jones, and R.J.Lipton. Secure databases: Protection against user influence. *ACM TODS*, 4(1):76–96, 1979.

[18] Y. Du, T. Xia, Y. Tao, D. Zhang, and F. Zhu. On multidimensional k-anonymity with local recoding generalization. In *ICDE*, pages 1422–1424, 2007.

[19] F.Chin. Security problems on inference control for sum, max, and min queries. *J.ACM*, 33(3):451–464, 1986.

[20] Jiri Fridrich, Miroslav Goljan, and Rui Du. Invertible authentication watermark for jpeg images. *Information Technology: Coding and Computing, International Conference on*, 0:0223, 2001.

[21] G.Aggarwal, T.Feder, K.Kenthapadi, R.Motwani, R.Panigrahy, D.Thomas, and A.Zhu. k-anonymity: Algorithms and hardness. *Technical report, Stanford University*, 2004.

[22] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. *Journal of Computer and System Science*, 60(3):592–629, 2000.

[23] G.Miklau and D.Suciu. A formal analysis of information disclosure in data exchange. In *SIGMOD*, pages 575–586, 2004.

[24] G.T.Duncan and S.E.Feinberg. Obtaining information while preserving privacy: A markov perturbation method for tabular data. In *Joint Statistical Meetings*. Anaheim,CA, 1997.

[25] Hakan Hacigümüş, Bala Iyer, Chen Li, and Sharad Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 216–227, New York, NY, USA, 2002. ACM.

[26] Hakan Hacigumus, Bala Iyer, and Sharad Mehrotra. Providing database as a service. In *ICDE '02: Proceedings of the 18th International Conference on Data Engineering*. IEEE Computer Society, 2002.

[27] I.P.Fellegi. On the question of statistical confidentiality. *Journal of the American Statistical Association*, 67(337):7–18, 1993.

[28] J.Byun and E.Bertino. Micro-views, or on how to protect privacy while enhancing data usability: concepts and challenges. *SIGMOD Record*, 35(1):9–13, 2006.

[29] J.Kleinberg, C.Papadimitriou, and P.Raghavan. Auditing boolean attributes. In *PODS*, pages 86–91, 2000.

[30] J.Schlorer. Identification and retrieval of personal records from a statistical bank. In *Methods Info. Med.*, pages 7–13, 1975.

[31] K.Kenthapadi, N.Mishra, and K.Nissim. Simulatable auditing. In *PODS*, pages 118–127, 2005.

[32] K.LeFevre, D.DeWitt, and R.Ramakrishnan. Incognito: Efficient fulldomain k-anonymity. In *SIGMOD*, pages 49–60, 2005.

[33] L.H.Cox. Solving confidentiality protection problems in tabulations using network optimization: A network model for cell suppression in the u.s. economic censuses. In *Proceedings of the Internatinal Seminar on Statistical Confidentiality*, 1982.

[34] L.H.Cox. New results in disclosure avoidance for tabulations. In *International Statistical Institute Proceedings*, pages 83–84, 1987.

[35] L.H.Cox. Suppression, methodology and statistical disclosure control. *J. of the American Statistical Association*, pages 377– 385, 1995.

[36] Feifei Li, Marios Hadjieleftheriou, George Kollios, and Leonid Reyzin. Dynamic authenticated index structures for outsourced databases. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 121–132. ACM, 2006.

[37] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *ICDE*, pages 106–115, 2007.

[38] Yingjiu Li, Huiping Guo, and Sushil Jajodia. Tamper detection and localization for categorical data using fragile watermarks. In *DRM '04: Proceedings of the 4th ACM Workshop on Digital Rights Management*, New York, NY, USA, 2004. ACM.

[39] Yingjiu Li, Vipin Swarup, and Sushil Jajodia. Constructing a virtual primary key for fingerprinting relational data. In *DRM '03: Proceedings of the 3rd ACM Workshop on Digital Rights Management*, pages 133–141. ACM, 2003.

[40] Eugene T. Lin and Edward J. Delp. A review of fragile image watermarks. In *Proceedings of the Multimedia and Security Workshop (ACM Multimedia '99)*, pages 25–29, 1999.

[41] Chun-Shien Lu, Hong-Yuan Mark Liao, and Liang-Hua Chen. Multipurpose audio watermarking. *Pattern Recognition, International Conference on*, 3:3286, 2000.

[42] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. L-diversity: Privacy beyond k-anonymity. *ACM Trans. Knowl. Discov. Data*, 1(1):3, 2007.

[43] Alfred J. Menezes, Paul C. Van Oorschot, Scott A. Vanstone, and R. L. Rivest. Handbook of applied cryptography, 1997.

[44] Ralph C. Merkle. A certified digital signature. In *CRYPTO '89: Proceedings on Advances in cryptology*, pages 218–238. Springer-Verlag New York, Inc, 1989.

[45] A. Meyerson and R. Williams. On the complexity of optimal k-anonymity. In *ACM PODS*, pages 223–228, 2004.

[46] Kyriakos Mouratidis, Dimitris Sacharidis, and HweeHwa Pang. Partially materialized digest scheme: an efficient verification method for outsourced databases. *The VLDB Journal*, 18(1):363–381, 2009.

[47] Einar Mykletun, Maithili Narasimha, and Gene Tsudik. Authentication and integrity in outsourced databases. *ACM Transactions on Storage*, 2(2):107–138, 2006.

[48] Maithili Narasimha and Gene Tsudik. Dsac: Integrity for outsourced databases with signature aggregation and chaining. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 235–236. ACM, 2005.

[49] N.R.Adam and J.C.Wortmann. Security-control methods for statistical databases: A comparative study. *ACM Comput. Surv.*, 21(4):515–556, 1989.

[50] HweeHwa Pang, Arpit Jain, Krithi Ramamritham, and Kian-Lee Tan. Verifying completeness of relational query results in data publishing. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, pages 407–418. ACM, 2005.

[51] HweeHwa Pang and Kian-Lee Tan. Authenticating query results in edge computing. In *ICDE '04: Proceedings of the 20th International Conference on Data Engineering*, page 560, Washington, DC, USA, 2004. IEEE Computer Society.

[52] P.Diaconis and B.Sturmfels. Algebraic algorithms for sampling from conditional distributions. *Annals of Statistics*, 26:363–397, 1995.

[53] Li Qiming and Chang Ee-Chien. Zero-knowledge watermark detection resistant to ambiguity attacks. In *MM&Sec '06: Proceedings of the 8th Workshop on Multimedia and Security*, pages 158–163. ACM, 2006.

[54] Sion Radu. Secure data outsourcing. In *VLDB '07: Proceedings of the 33rd International Conference on Very Large Data Bases*, pages 1431–1432. VLDB Endowment, 2007.

[55] R.J.Bayardo and R.Agrawal. Data privacy through optimal k-anonymization. In *ICDE*, pages 217–228, 2005.

[56] P. Samarati. Protecting respondents' identities in microdata release. *IEEE Trans. on Knowl. and Data Eng.*, 13(6):1010–1027, 2001.

[57] S.Chawla, C.Dwork, F.McSherry, A.Smith, and H.Wee. Toward privacy in public databases. In *Theory of Cryptography Conference*, 2005.

[58] Mok-Kong Shen. Algorithm 202: generation of permutations in lexicographical order. *Commun. ACM*, 6(9):517, 1963.

[59] Radu Sion. Query execution assurance for outsourced databases. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 601–612. VLDB Endowment, 2005.

[60] Radu Sion, Mikhail Atallah, and Sunil Prabhakar. Rights protection for relational data. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 98–109. ACM, 2003.

[61] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *SP '00: Proceedings of the 2000 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2000.

[62] T.Dalenius and S.Reiss. Data swapping: A technique for disclosure control. *Journal of Statistical Planning and Inference*, 6:73–85, 1982.

[63] Juan Ramón Troncoso-Pastoriza and Fernando Pérez-González. Zero-knowledge watermark detector robust to sensitivity attacks. In *MM&Sec '06: Proceedings of the 8th Workshop on Multimedia and Security*. ACM, 2006.

[64] V.I.levenshtein. Binary codes capable of correcting deletions, insertions and reversals. In *Soviet Physics Doklady 10*, pages 707–710, 1966.

[65] Terry A. Welch. A technique for high-performance data compression. *Computer*, 17(6):8–19, 1984.

[66] R.C. Wong, A.W. Fu, K. Wang, and J. Pei. Minimality attack in privacy preserving data publishing. In *VLDB*, pages 543–554, 2007.

[67] R.C. Wong, J. Li, A. Fu, and K. Wang. alpha-k-anonymity: An enhanced k-anonymity model for privacy-preserving data publishing. In *KDD*, pages 754–759, 2006.

[68] Min Xin, Haixun Wang, Jian Yin, and Xiaofeng Meng. Integrity auditing of outsourced data. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 782–793. VLDB Endowment, 2007.

[69] Min Xin, Haixun Wang, Jian Yin, and Xiaofeng Meng. Providing freshness guarantees for outsourced databases. In *EDBT '08: Proceedings of the 11th international conference on Extending database technology*, pages 323–332. ACM, 2008.

[70] X.Xiao and Y.Tao. Personalized privacy preservation. In *SIGMOD*, pages 229–240, 2006.

[71] Yin Yang, Dimitris Papadias, Stavros Papadopoulos, and Panos Kalnis. Authenticated join processing in outsourced databases. In *SIGMOD '09: Proceedings of the 35th SIGMOD international conference on Management of data*, pages 5–18. ACM, 2009.

[72] Chia-Mu Yu and Chun-Shien Lu. Robust non-interactive zero-knowledge watermarking scheme against cheating prover. In *MM&Sec '05: Proceedings of the 7th Workshop on Multimedia and Security*. ACM, 2005.

[73] L. Zhang, S. Jajodia, and A. Brodsky. Information disclosure under realistic assumptions: privacy versus optimality. In *CCS*, pages 573–583, 2007.

[74] L. Zhang, L. Wang, S. Jajodia, and A. Brodsky. Exclusive strategy for generalization algorithms in micro-data disclosure. In *Proceeedings of the 22nd annual IFIP WG 11.3 working conference on Data and Applications Security*, pages 190–204, 2008.