Exploring personalized life cycle policies

CTIT Technical Report

H.J.W. van Heerde* N. Anciaux† M.M. Fokkinga* P.M.G Apers*

December, 2007

Abstract

Ambient Intelligence imposes many challenges in protecting people's privacy. Storing privacy-sensitive data during for permanently will inevitably result in privacy violations. Limited retention techniques might prove useful in order to limit the risks of unwanted and irreversible disclosure of privacy-sensitive data. To overcome the rigidness of simple limited retention policies, Life-Cycle policies more precisely describe when and how data could be first degraded and finally be destroyed. This allows users themselves to determine an adequate compromise between privacy and data retention. However, implementing and enforcing these policies is a difficult problem. Traditional databases are not designed or optimized for deleting data. In this report, we recall the formerly introduced life cycle policy model and the already developed techniques for handling a single collective policy for all data in a relational database management system. We identify the problems raised by loosening this single policy constraint and propose preliminary techniques for concurrently handling multiple policies in one data store. The main technical consequence for the storage structure is, that when allowing multiple policies, the degradation order of tuples will not always be equal to the insert order anymore. Apart from the technical aspects, we show that personalizing the policies introduces some inference breaches which have to be further investigated. To make such an investigation possible, we introduce a metric for privacy, which enables the possibility to compare the provided amount of privacy with the amount of privacy required by the policy.

1 Introduction

In many application domains, but with Ambient Intelligence and ubiquitous computing in particular, privacy sensitive data are collected and stored for future use, typically to fulfill one or more purposes, or to improve a certain service. A straightforward example is a web shop collecting address and credit card information in order to process the payments and the dispatch of the ordered goods. It is clear that without collecting these data, the service cannot be provided. However, a customer exposes himself or herself to a possible privacy threat since the data will be out of sight and control, stored 'somewhere

*CTIT, University of Twente, The Netherlands
†INRIA Ronquencourt, France

on the Internet'. The customer has to trust the web shop (including its database administrators) not to sell the information to third parties, and has to rely on the web shop's protection against hackers or other forms of abuse. Moreover, in cases where the privacy protection is breached, the customer might not notice it soon enough, so that it might no longer be traceable once the violation comes to light. This principle is know as *information asymmetry*, where the data owners (the customers) have less knowledge about the use of their data than the data collectors and data users [15].

What is already a problem with data collection on the Internet, will eventually become an even larger problem with ubiquitous computing. In an ambient intelligence environment, sensors are predicted tol continuously monitor people everywhere during their daily lives. Without developing adequate privacy improving techniques, it will not be clear for the donor—we call the subject being monitored the *donor* of *his* or *her* data—which data is stored where for how long and who has access to it. Moreover, after the data has been collected, the donor no longer has control over it. Those observations have lead to two particular proposals to enhance privacy, deduced from privacy laws [11] and the work on Hippocratic databases [2]:

Openness: make data collection and retention fully transparent to the donor; where will the donor be monitored, and which data is currently retained by the collector in what form?

Control: give the donor the ability to control his data; which data can be retained and who has access to it?

Without the guarantee of having *openness* and *control* of his or her data, the privacy aware donor might stop donating data at all, eventually resulting in futile services. A donor who has insight in his or her own data, controlling that only not too privacy sensitive data is kept, might present just enough data for a service to be sufficiently effective for that donor.

Both openness and control are good tools to at least give the donors the opportunity to protect their own privacy (see Example 1 for a practical example). Still, by only offering openness and control, donors have to act *themselves*, shifting the responsibility of good privacy protection back to the donor. A straightforwarded first step to *simplify* the control over which data must be removed from the history is to automatically remove the data after a predefined retention period. This can be specified in a user-defined policy, or by organization-wide policies. The data collector is then responsible for putting such policies into effect and enforcing them [2].

However, simply removing data after a certain retention period may be too rigorous to achieve an optimum balance between *smartness* and *privacy*, so an intermediate solution may be desirable. In previous works [4], we presented a model for *life-cycle policies* (LCP) which specify a step-wise per-attribute destruction of data and investigated how our model could be enforced using traditional databases. For example, a life cycle policy can define that a location will be stored accurately as a set of *coordinates* for only 10 minutes. After this period, a transition takes place, degrading the accuracy of the location to *building*. Again, after a week, the location is degraded to *city* until final removal of the value altogether (see Figure 1 for a graphical representation).

Google has acknowledged that it stores users' search terms for a undefined period, even when users haven't given explicitly consent [14] to do so. Ordinary users are not aware of being monitored, and even if they are aware, they don't know exactly what their search history exactly contains. A search history might even be *polluted* with search terms which aren't semantically or logically related to the user. Hence, there is a total lack of *openness*, and as a direct consequence, *control*.

Google introduced a new service which *explicitly* stores a user's search history, and only when the user has subscribed to this service. Through a web interface users have full insight into their search history. Moreover, they can delete items from their history. Hence, *openness* and *control* give users the ability to influence which data will be retained, being aware of the risk that at least only this data can be subject to privacy violation. In this way, Google lets the users themselves decide to which extent they want to provide data in order to improve the quality of service that Google can provide them.

Example 1: Google Web History [24], a new service introduced to personalize search results and to give the user the ability to browse through his search history.

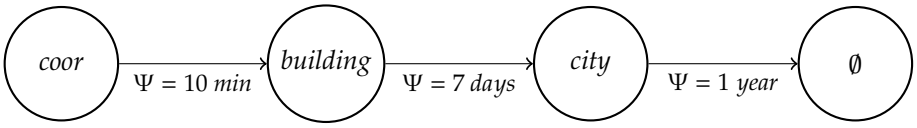


Figure 1: *Graphical representation of a simple life cycle policy for a location attribute. Edges denote transitions between states of accuracy after a retention period* Ψ*, from coordinates to building until final destruction of the data denoted by* $\emptyset$*.*

1.1 Contribution

The techniques developed for managing the degradation of data [4] are based on the simplifying assumption that the stored data is subject to only one life-cycle policy. Although such a policy can represent the mutual privacy wishes of a certain group of donors, this approach is too limited in terms of matching the privacy preferences of all kind of donors, ranging from the paranoid to the trustful donor. In this report, we extend the model such that each donor can specify his or her own policy, in order to give the donor more control about his or her own data. We term these policies *personalized* life cycle policies. In this report we explore the following research questions, which we will discuss in more detail throughout the document. Note however that the target of this technical report is to open up a research agenda, not to provide final solutions to the stated problems.:

- Can the storage structure used for one policy easily be adopted to support

multiple policies and what might be the impact on performance?

- What kind of *inference breaches* arise when managing multiple policies?
- Can the level of *required* and provided privacy be captured using a metric, and can such a metric point out how effective a chosen policy is compared to the policies of other donors?

In Section 2 we will first recall the necessary definitions of the Life-Cycle Policy model. Than we continue in Section 3 with describing the storage structure techniques used to manage only one collective policy, we will investigate the consequences of managing multiple policies in terms of technical challenges and propose preliminary solutions. When the technical problems are resolved, we show inference breaches which are introduced with personalized policies in Section 4 . Finally, we propose a metric to capture the amount of privacy required and provided by a policy. We start however with some background of related work on privacy techniques.

1.2 Related work

A first attempt to express privacy policies based on regulations, has been conducted by the Platform for Privacy Preferences, known as P3P policies [25]. These policies let users know which data will be collected for what purpose, and how long the data will be retained. Although P3P is supported by several modern web browsers, and many web sites already specify policies, the policies are quite concealed and few users actually read them or are able to fully understand them [8]. Indeed, there are tools available nowadays to express user preferences which can be matched against the collectors' policies [3]. Still, P3P only *describes* policies and does not *enforce* them, making P3P little more than a standardized complement to the privacy laws of most countries [12]. Nevertheless, P3P has been a first step in making the handling of privacy sensitive data more transparent, increasing the information symmetry and putting users back in control.

Building on P3P, techniques such as the privacy aware database (PawS) have been developed, letting the system automatically interpret and apply the policies to the data [16]. Other frameworks also exist which enable users to specify personal location privacy policies [21], letting users decide on the accuracy of the data disclosed to service providers. Furthermore, the work on Hippocratic databases [2] has been inspired by the principle that databases should be responsible for the privacy preservation of the data they manage. Hippocratic databases are founded on some key principles which have their roots in privacy laws. One of the principles is the *limited disclosure* principle, which states that data may not be disclosed unless this complies with the specified purposes for which consent has been given by the donor. Implementation frameworks behind such a system, based on *access control* mechanisms, already exist [2, 5]. However, such systems are still based on trust [1]; trust which cannot be put forever on a system. Even when secure access control techniques are used, a database administrator can be or become malicious. Even if the chosen security regime can be proven successful now, it might be not in the future [9].

Limited retention techniques have been proposed to ensure that data can no longer be subject to occasional disclosure. The limited retention principle is

a key principle behind many privacy laws and p3p, and has also been adopted in the work on Hippocratic databases. Still, the "all-or-nothing" behavior of limited data retention leads to overstatement of the retention limit. The retention period is often based on a different purpose from that for which the data has been collected and stored in the database. This implies that the data will be retained as long as needed for the longest lasting purpose, whereas for the shorter-term purposes could have been fulfilled with less data. Moreover, regarding databases storing this data, even ensuring that data is irreversibly removed from the system is not a straightforward task [4, 19, 22].

In addition to access control, security measures for protecting a database server, such as data encryption, firewalls, and intrusion detection systems can be used. Those techniques make attacks more difficult without completely preventing them. Recent studies have shown concerns about weaknesses in widely used encryption mechanisms and even suspect governmental organizations for deliberately inserting flaws in algorithms they propagate to use in security systems [20]. Intrusion detection systems [7] are especially useful against repetitive attacks such as spying on a database, although it is still hard to find a good balance between false negative and false positive detections. However, used in addition to data degradation, IDS would make it very hard for even a determined attacker to obtain a large consecutive history of accurate data.

While limiting the risks of attacks is good practice, data can still be subpoenaed by a Court and may then be subjected to forensic analysis. New business alliances might increase the sensitivity of the data by merging the data with newly obtained datasets for which the old privacy policies are no longer adequate. For these reasons, proposals have been made to make the donor him or herself responsible for protecting his or her own data, not just relying on the philosophy of trusting organisations to protect his privacy. In their vision paper, Aggarwal et al proposed the P4P framework [1], in which the donor keeps control about which information to release to service providers. They consider the 'paranoid' user who doesn't trust the collecting organizations, in contrast to the users of P3P frameworks. Related to this are client-based encryption techniques, in which the service provider is unable to decrypt the data at the server [13]. Here the user is needed in order to obtain access to the data, placing the user in a privacy protection role. Although these solutions are robust against server attacks, the accessibility for service providers is much lower, leading to high communication costs when data needs to be queried or updated and placing constraints on how applications are developed and deployed. However, data degradation doesn't place these restrictions on applications, data can still be stored at the server side and by enforcing data degradation, donors are in control of the level of privacy risk they want to take in terms of retention periods.

1.2.1 Anonymization techniques

Anonymization of data might be a solution to prevent disclosure of privacy sensitive-data. In fact, major companies such as Google already state they will adopt anonymization as a measure to improve privacy protection [10]. *k*-Anonymity [23] is based on the idea of masking (parts of) the (quasi) identifier of a partly privacy-sensitive tuple, such that the sensitive part of the tuple will

be hidden between $k-1$ potential identifier candidates within the same dataset. For example, the zip-code, date of birth and gender may uniquely identify an individual and reveal the corresponding sensitive data. By masking the date of birth, the dataset should contain at least k occurrences of the same zip-code, gender combination. The work on l-diversity [17] goes a step further by taking background knowledge into account, enforcing enough diversity between the privacy sensitive attributes.

Usually, anonymization is applied to large datasets at once, making sure that for each tuple, the tuple shares the same identifier with $k-1$ others. In practice this could result in a strictly k-anonymous database at the cost of losing much usability. Although Byun et al provided a technique to update anonymized databases [6], each time new data arrives, the database has to be sanitized into a k-anonymous state again, making it hard to obtain a clear view of the database from an application perspective, since old tuples might be sanitized at unpredictable times. Moreover, given the additional values of the newly inserted data, old data might be too strictly anonymized in terms of loss of usability given the new dataset. The latter can only be solved by maintaining information about previous states, which in terms of privacy requirements is undesirable. Besides, correctly anonymizing the data is a hard problem [18]. To illustrate, a good example of incorrect and insufficient use of anonymization has been given when American Online decided to put a large set of search queries online [14]. AOL anonymized the IP addresses of the computers from which the queries were issued, which was not enough to prevent attackers from inferring many privacy sensitive facts.

1.2.2 Data degradation

Data degradation, as in our life cycle policy model, can be complementary to all discussed techniques. Firstly, by limiting the impact of inevitable privacy breaches, data degradation is complementary to access control, since data which has been subject to degradation either has a lower level of accuracy and thus sensitivity, or has already been removed from the system. Moreover, although only on temporary basis, accurate data can still be protected against regular attacks with the use of access control techniques. Secondly, anonymization is good practice when datasets have to be made public without revealing too much sensitive data; for example, when used for disclosing datasets for research purposes, and therefore it can be a complementary technique to data degradation. Data degradation is particularly useful when data needs to be accurate for some time to make well-defined services possible, requiring a certain amount of (uniform) data accuracy. Moreover, a degradation model can keep the identifier of the donor intact; hence, user-oriented services can still exploit the information to the benefit of the donor.

2 The Life-Cycle Policy model

In our data degradation model, termed the *life cycle policy model*, data is subject to a progressive degradation from the accurate state to less detailed intermediate states, up to disappearance from the database. The degradation of each piece of information (typically an attribute) is captured by a *Generalization Tree*. Given

a *domain generalization hierarchy* for an attribute, a generalization tree (GT) for that attribute gives, at various levels of accuracy, the values that the attribute can take during its lifetime. Hence, a path from a particular node to the root of the GT expresses all degraded forms the value of that node can take in its domain. Furthermore, for simplicity we assume that for each domain there is only one GT.

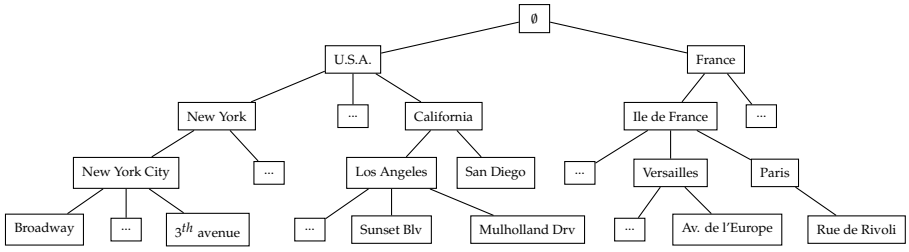


Figure 2: Example of a generalization tree for the location attribute. The leafs of the tree denote the most accurate values (addresses).

A *life cycle policy* (LCP) governs the degradation process by fixing how attribute values navigate from the GT leaf up to the root. While we may consider complex life cycle policies where state transitions are triggered by events, we make the simplification that policies express degradation triggered by time. A LCP for an attribute is modeled by a deterministic finite automaton as a set of degradable attribute states $\{d_0, \ldots, d_n\}$ denoting the levels of accuracy of the corresponding attribute d, a set of transitions between those states and the associated time delays (Ψ_i) after which these transitions are triggered. The time between transitions is denoted by Δ_k, hence $\Psi_k = \sum_{i=0}^{k} \Delta_k$.

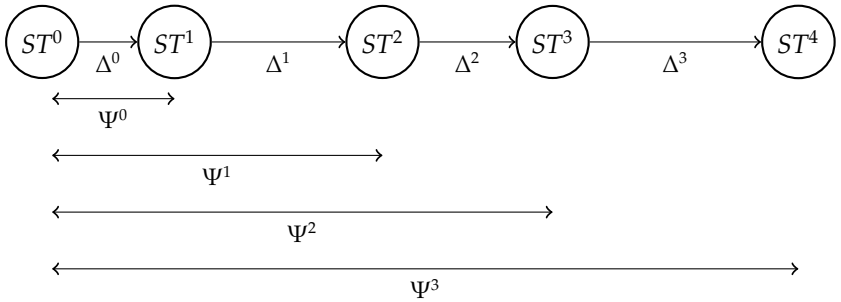


Figure 3: *Representation of the life cycle of a tuple. The tuple will be first inserted into ST_0. After a period Δ_0, it will be member of ST_1. After Δ_1 since Δ_0 (at Ψ_1 after insertion time), it will be member of ST_2, et cetera.*

We redefine a *tuple* as a composition of stable attributes—which do not participate in the degradation process—and degradable attributes. The combination of LCPs of all degradable attributes makes that, at each independent attribute transition, the tuple as a whole reaches a new tuple state t_k, until all degradable attributes have reached their final state. A tuple LCP is thus derived from the combination of each individual attributes' LCP.

Due to degradation, the dataset DS is divided into subsets ST_k of tuples within the same tuple state t_k, having a strong impact on the selection and projection operators of queries. These operators have to take accuracy into account, and have to return a coherent and well-defined result. To achieve this goal, data subject to a predicate P expressed on a demanded accuracy level k, will be degraded before evaluating P, using a degradation function f_k (based on the generalization tree(s)). Given f,P and k, we define the select and project operators $\sigma_{P,k}$ and $\pi_{*,k}$ as:

$$\sigma_{*,k} = \sigma_P\left(f_k\left(\bigcup_{i=0}^{k} ST_i\right)\right) \qquad \pi_{*,k} = \pi_*\left(f_k\left(\bigcup_{i=0}^{k} ST_i\right)\right)$$

The accuracy level k is chosen such that it reflects the declared purpose for querying the data. Then, queries can be expressed with no change on the SQL syntax in the example below:

```
DECLARE PURPOSE Stat
SET ACCURACY LEVEL Country for location, Range1000 for salary

SELECT * FROM Person
WHERE location like 'France' and salary = '2000-3000'
```

The semantics of update queries is as follows. Firstly, the delete query semantics is unchanged compared to a traditional database, except for the selection predicates which are evaluated as explained above. Hence, the delete semantics is similar to the deletion through SQL views. When a tuple must be deleted, both stable and degradable attributes will be deleted. Secondly, insertions of new elements are granted only in the most accurate state. Finally, we make the assumption that updates of degradable attributes are not granted after the tuple creation has been committed. On the other hand, updates of stable attributes are managed as in a traditional database.

2.1 Personalized policies

In comparison to our previous work [4], we release the constraint on the model of having only one single LCP. Hence, within this model, multiple life cycle policies can be defined handling the same set of attributes. Still, we make the simplification that a user can only define one new policy, or it can share a policy with other users. Moreover, we apply the following simplifying restriction:

- All policies follow the same *degradation pattern*, that is, all policies have the same set of tuple states (although all policies can have distinct retention periods Δ).

We define the maximum retention period of all n policies Ψ^k_{max} as $max(\Psi^k_0, \ldots, \Psi^k_n)$. See for an example Figure 4.

3 Storage structure

An important question is whether data degradation can be implemented in a database management system. Traditional databases are not optimized or

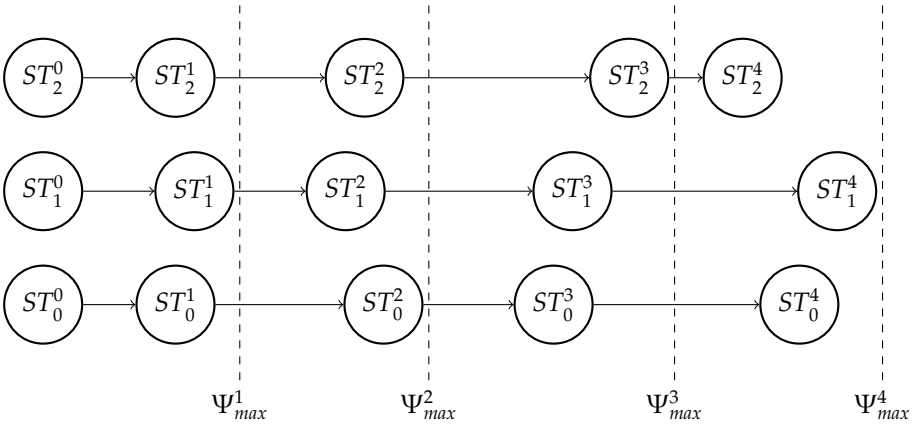


Figure 4: *Multiple policies where ST_i^k denotes tuples in state k subject to policy i.*

designed to physically remove data. A regular delete operation doesn't guarantee that data is physically removed from the disk, such that it cannot be accessed anymore [19, 22]. New techniques for storage structures, indexes and log management must be designed in order to guarantee irreversible destruction of the data, while keeping reasonable performance. In this section, we recall some of the storage techniques we proposed earlier [4]. As mentioned before, those techniques are based on the assumption that only one single policy has to be managed for all data. We will show where those techniques rely on this assumption, we identify what the consequences are of handling multiple personalized policies, and propose possible extensions to the storage structure.

3.1 Single policy management

Suppose that for each attribute there is only one single life-cycle policy. Then the following important property holds for tuples x stored in a relation R: since there is only one policy for each attribute, the order of degradation is perfectly determined by the order of insertion, or more precisely:

$$\forall x : R, x' : R \mid x.T_{insert} < x'.T_{insert} \Rightarrow x.T_{degrade} < x'.T_{degrade} \quad (1)$$

Tuples d are subject to degradation, where T_{insert} is the time a tuple is inserted into R and $T_{degrade} = T_{insert} + \Psi$ the time the tuple must be degraded, where Ψ is the retention period for that attribute as specified in the policy. The δ-*timeliness* assumption states that degradation can take place within the interval $[T.degrade - \delta, T.degrade]$ (where δ can be defined as a small fraction ρ of the retention period Ψ). Together those properties imply that groups of tuples which are inserted within a certain interval of length δ can also be degraded together.

To store (and remove) data efficiently, the cost of a random I/O operation must be shared with as many tuples as possible. An obvious solution is to *buffer* inserts in *RAM* until the buffer is full and the tuples are flushed to the disk, consuming only one random I/O operation (and perhaps some cheap sequential I/O operations depending on the size of the buffer). In this way, full

pages can be written to disk at once. Because of the natural ordering of tuples on degradation time, tuples which have been inserted within the same time interval of size δ can also be *degraded* together using only one single random I/O operation. Those tuples are fetched together into a *degradation buffer* in which the actual degradation takes place. This is shown in Figure 5.

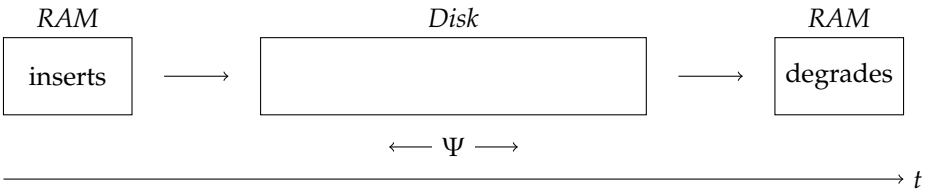


Figure 5: *Tuples are 'sorted' on both insert and degradation time, making both sequential inserts and updates possible.*

A degradation schedule gives the times when the data has to be degraded. This can simply be implemented by maintaining a table in which pointers to the pages to be degraded are stored with the corresponding degradation time. Since tuples are grouped by intervals of size δ, time can be measured in units of δ, so that $t = n$ gives the time of the n^{th} degradation step.

Figure 6 shows the logical representation of a file containing the pages with tuples. With each new insert the offset of a page in the file increases (where the offset represents the address of the page on disk, although this might not be physically true due to fragmentation), offsets of tuples within a page start with the oldest tuple in the page (hence there are gaps within the offsets).

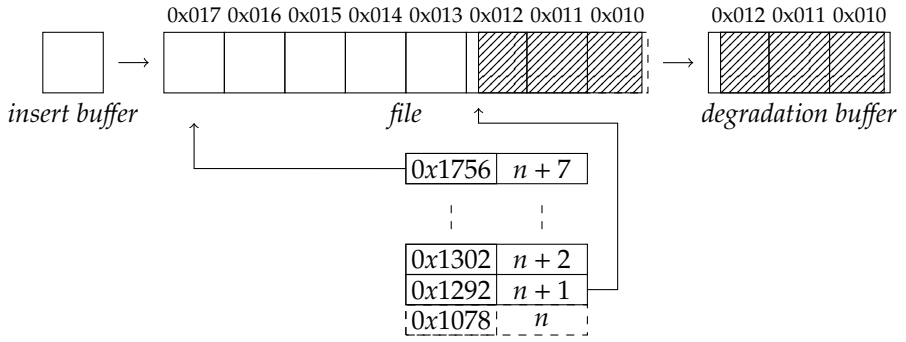


Figure 6: *Degradation schedule responsible for degradation. The blocks represent full pages, the shaded area represents tuples which can be degraded together. The page with offset* $0x10$ *has already been moved to the degradation buffer at degradation step* n*, since a part of the tuples had to be degraded in that step. For degradation step* $n + 1$*, a part of the tuples of page* $0x10$ *are thus already in the buffer (and therefor not on disk anymore). To complete the degradation, pages* $0x11$ *and* $0x12$ *will also be placed in the buffer. Offsets in the buffer correspond to addresses of the pages when they were still on the disk.*

In traditional data structures, tuples are stored as a single data item on disk. In this way, a fixed sized page consists of a set of full tuples. When an attribute's

policy dictates that a particular attribute value of a set of tuples needs to be degraded (replaced by a less accurate value) or destroyed, the page containing the tuple needs to be accessed such that the single attribute of the tuples can be overwritten. As a result, full tuples are fetched while only a part of each tuple will be affected. Although this storage model is efficient for queries where full tuples need to be fetched to construct the result, this is not efficient for degradation. To speed up degradation, tuples can be fragmented such that all the attributes of a tuple are stored in separated pages. Therefore, more attributes can be stored in one page, making accessing a page fully efficient since only data which need to be degraded will be fetched.

Degrading an attribute to a less accurate value can be achieved by updating the tuple, and replacing the old accurate value with the new less accurate value. During an update operation, the page containing the data item needs to be read into RAM to know with which value the old value needs to be replaced, and be written back to disk. This makes the operation inherently more expensive than delete operations, where a page only needs to be overwritten. By precomputing all attribute states and storing the attributes with all different states of accuracy in separate files, degradation can be achieved by only delete operations (see Figure 7). Indeed, degradation performance can be increased with the cost of introducing redundancy, making insert operations less efficient.

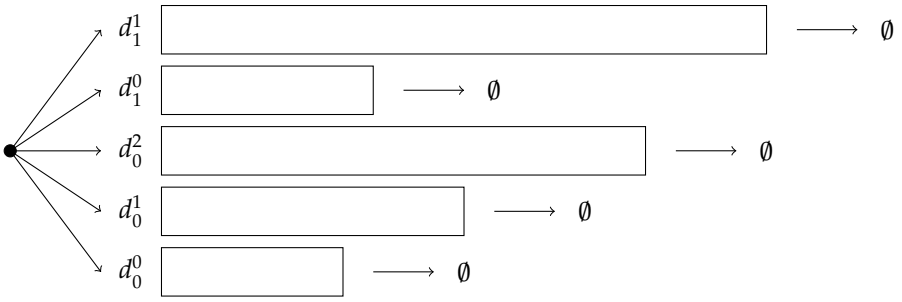


Figure 7: *Sequential writes and deletes using the fragmented/eager storage strategy.*

3.2 Multiple policies management

In this section we discuss the implementation and performance problems which may arise when introducing multiple policies. For the restrictions on this new model we refer to section 2.1.

When we introduce multiple policies in one data store, we lose the property that the degradation order is determined by the insertion order. Indeed, if $x.T_{insert} > x'.T_{insert} \wedge x'.\Psi - x.\Psi > x.T_{insert} - x'.T_{insert}$ then $x'.T_{degrade} > x.T_{degrade}$. It is clear that sequential degradation is not possible when the tuples/attributes stored in one page have different degradation times. Moreover, the degradation schedule has to keep track of each individual tuple and the time it has to degrade. Also, in terms of storage space (and therefore also of query performance where sequential scans are involved) the cost is higher, since a page will only be partially used after some part of it has been degraded. It will take longer

until space is free for new pages [1].

One solution is to create a one-page-buffer dedicated to tuples belonging to the same policy, waiting to be populated by tuples which can be degraded within the same interval. Thus, tuples sharing the same retention period Ψ_i inserted within the interval $[n\delta, (n+1)\delta]$, will be read into the same insert buffer. However, effectivness mainly depends on δ, the insertion rate and the numer of policies. If δ is small, pages will not be filled before they must be appended to the file; if the insertion rate is too low and if there are many different policies (for example: if 200 tuples fit into one page, $\Psi = 10$ minutes, $\delta = \rho\Psi = 0.01\Psi = 6$ seconds, and there are 10 distinct policies, then the insertion rate must at least be $\frac{200}{6} \times 10 = 333$ inserts per second).

Using insert buffers to group tuples with almost equal times of degradation (within the interval of δ), the degradation phase can still benefit from the fact that a random I/O operation is shared by all tuples of a page. However, since there is no longer any ordering on degradation time between pages, *fewer* pages (most probably only one) can be moved sequentially to the degradation buffer, making degradation less efficient. Moreover, because pages are not ordered according to degradation time, gaps will occur in the file, making querying using sequential scans less efficient. This is shown in Figure 8.
There are several preliminary options to optimize degradation of data subject to different policies, which all can be investigated in terms of the additional amount of RAM needed and the performance loss compared to the structure without multiple policies. For example:

- Maintain a set of insert buckets in which tuples will be placed. Those tuples will be degraded at the same time, based on the insert time and their Ψ. For example, a tuple with $\Psi = 10$ and $T_{insert} = 0$ will degrade at the same time ($T_{degrade}$) as a tuple with $\Psi = 5$ and $T_{insert} = 5$, and can be placed in the same bucket B. When $B.T_{degrade} < now() + \Psi_{min}$, where Ψ_{min} denotes the minimum retention period of all policies, the bucket can be closed and inserted into the file. In this way the file will be ordered by degradation time (Figure 9).

- Sequentially fetch a set of pages into a degradation buffer. This buffer will now contain pages which will not degrade at the same time. Degrade only those pages that can be degraded given their degradation time, and keep the others. Replace the degraded pages with pages from the file, and again degrade the pages that can be degraded given their degradation time.

  How effective this strategy is depends on the number of different policies. Special care has to be taken in a situation, in which the degradation buffer is completely filled with undegradable pages due to longer retention periods. This might 'block' other pages with shorter retention periods.

Those proposals are not meant to be exhaustive and must be seen as a first analysis of the problems which arise when dropping the single policy constraint. Besides, a performance analysis of these options is planned as future work. Moreover, an thourough investigation of other import technical

[1] Although not discussed yet, when pages can be degraded at once, this means that in some occasions it is possible to overwrite the page with a new page, making deletion more efficient.

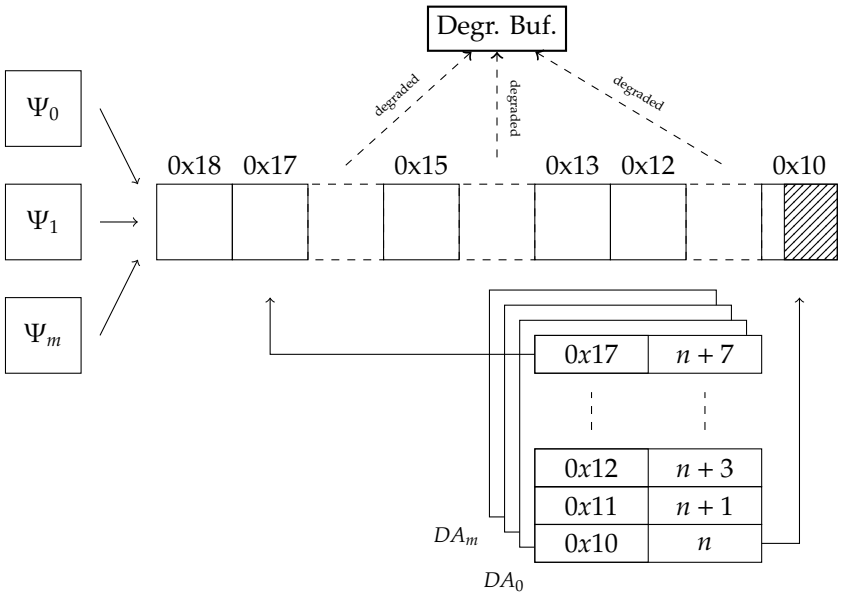


Figure 8: *Pages are not ordered on degradation time, making it not possible anymore to sequentially degrade multiple pages, leaving gaps in the file. Pages are filled using insert buffers during a period of length δ, when a page is full, or at the end of the period, it will be appended to the file. For each distinct Ψ a degradation schedule (DA) will be maintained. Each entry in DA_i refers to an offset of a page inserted in $[\Psi_i + n\delta, \Psi_i + (n+1)\delta]$. Example: the shaded tuples in the only partially filled page $0x10$ will be degraded at degradation step $\Psi_0 + n$. Tuples in pages with offset $0x11, 0x14$ and $0x16$ already have been degraded using the degradation buffer.*

aspects like the indexes, transaction management and log management have to be performed in order to oversee all technical consequences.

4 Inference breaches

In this section we investigate which privacy breaches will be introduced when managing multiple policies. Since different data items can be subject to different policies, inference breaches exist due to added knowledge that there are different policies. We will see that the amount of knowledge about the policies will determine how much information can be resolved by analyzing the not yet degraded data. First we introduce some basic concepts defining the amount of privacy we can expect based on our model.

4.1 Definition of 'upgrade probability' and privacy violation

When an attribute value has been degraded to a less accurate value, we assume that this degradation was irreversible: we assume it is not possible to traverse down a generalization tree. However, when degrading an attribute, the new value will be related to the old value (otherwise the degradation loses all its

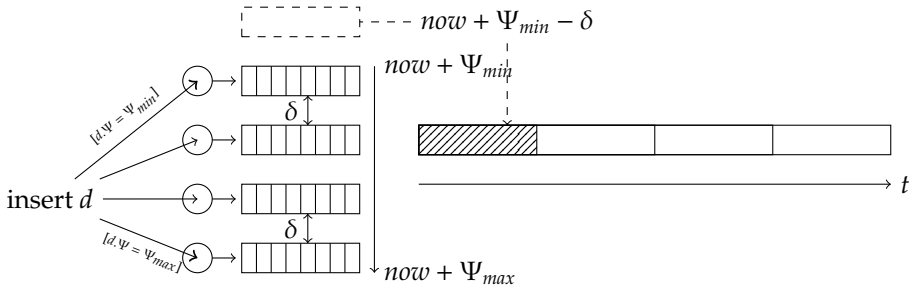


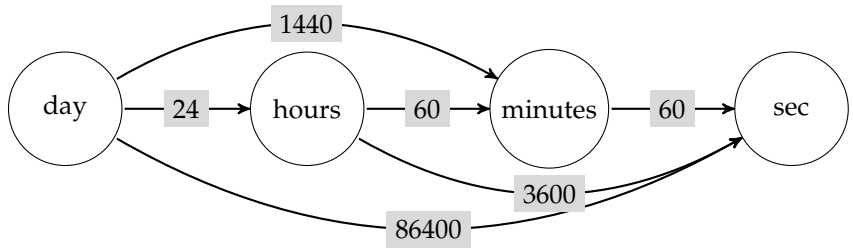
Figure 9: *Buckets are used to temporarily store pages which have tuples which will degrade in the same interval of size δ. At time now + $\Psi_{min} - \delta$, no tuples will arrive anymore which could be inserted in the bucket corresponding to degradation time now + Ψ_{min}, so that this bucket can be moved to the file which will remain ordered on degradation time. Using this strategy, exactly $\frac{\Psi_{max}-\Psi_{min}}{\delta}$ buckets have to be managed, of which the size depends on the insertion rate. Note that the buckets can be stored on disk, and flushed using only one random I/O to the final file.*

usability). This relation between old and new value will be shared among a limited set of values (hence, a limited set of persons will degrade to the same group value). There is always a possibility to guess the original value, to *upgrade* the degraded value to a more accurate state. The probability of guessing the *exact* original value will be called the *upgrade probability*.

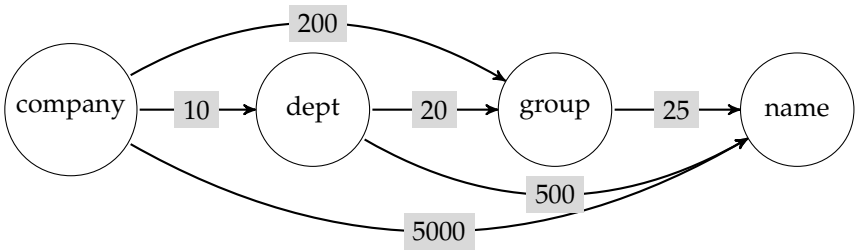
Let's consider k-anonymity [23]: the k stands for "a particular data item will be *hidden* between at least $k - 1$ others". This means that after *degrading* a particular item to a next state with k distinct values, the probability for correctly *upgrading* the item to the original value is $\frac{1}{k}$. Thus, if we want to hide a particular name within a group of 20 persons, an attacker can only retrieve with a probability of $\frac{1}{20}$ that the degraded identity belongs to that person. Analogously, when time in seconds has been degraded to day (a day consists of 86400 seconds), than the probability of guessing exactly the original time is $\frac{1}{86400}$.

It is obvious that in the case of time degradation the chance of guessing the time value is only $\frac{1}{86400}$, but the probability of guessing the original time value with an error variance of 30 seconds (e.g., guessing the minute) is much higher, $\frac{1}{1440}$, and for the original hour even $\frac{1}{24}$. Indeed, if you know the hour you already know much more than if you only know the day, and that 'only' with a probability of $\frac{1}{24}$. The same applies to the *id* domain.

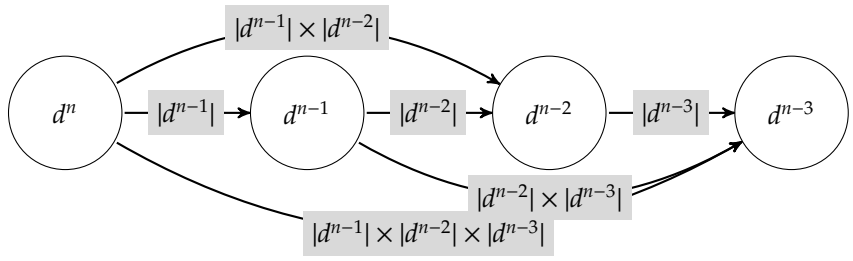
Figure 10 shows the domain generalization hierarchies (*DGH*) of time and id. The edges indicate the group size of the underlying accuracy levels. For example, if you know the exact hour, the total number of possible seconds within that hour is $|min| \times |seconds| = 3600$ and the number of possible minutes ($|min|$) is 60. If the current state is *dept*, the number of names within the know department is approximately 500 ($|group| \times |name|$). Speaking in terms of changes to upgrade at least *something*: if the current state is *day*, the probability for guessing at least the original hour, and perhaps also the original minute or the



(a) Time



(b) ID



(c) General

Figure 10: *Example (reversed) domain generalization hierarchies of the time and id domain. The labels at the edges denote the size of the groups (a day consists of 24 hours, 1440 minutes, et cetera). Figure c shows the generalization hierarchy of an attribute d.*

original second is (hence, when you don't know the hour you know nothing):

$$\begin{aligned} \textit{Upgrade robability(day)} &= 1 - \textit{(not even upgrade the original hour)} \\ &= 1 - \left(\frac{|hour| - 1}{|hour|}\right) \\ &= \frac{1}{|hour|} \end{aligned}$$

and in the more general form (X = *guess at least one original value given the degraded value*, n is the level of accuracy of the degraded value):

$$\begin{aligned} P_n(X) &= \textit{non-trivial upgrade probability from level } n \\ &= 1 - \left(\frac{|d^{n-1}| - 1}{|d^{n-1}|}\right) \\ &= \frac{1}{|d^{n-1}|} \end{aligned}$$

where $|d^{n-1}|$ is the *group size* of the $n-1^{th}$ generalization step assuming that we know what the parent value d^n is (hence, $P_n(X)$ gives us the probability that we upgrade a visible value to an already degraded value). By definition, we cannot decrease this upgrade probability given a certain policy. However, an attacker may try to *increase* the upgrade probability, trying to violate privacy. We speak of *privacy violation* when:

$$P_n(X) \quad > \quad \frac{1}{|d^{n-1}|}$$

In Section 5 we will try to capture the *amount* of privacy violation in a more quantitative metric. For the moment we are only interested in the case that *at least something* has been upgraded. Indeed, above definition of the probability to upgrade something doesn't say anything about *how much* the value has been upgraded. In the next section we will show that an attacker can violate privacy by simply looking to a snapshot of the database. We only give examples based on upgrading the identifier attribute. However, inference breaches based on other attribute (like time) exists. A further investigation on this topic is required in the future. In this technical report we only show that there indeed are inference breaches when introducing multiple policies.

4.2 Refining identity

A relation R has two degradable attributes *id* and *time* of which the generalization hierarchies are pictured in Figure 10. For notational purposes, we extend the set of attributes $\{id, time\}$ with al the states both attribute can take, and the corresponding retention period Ψ: $R(id^1, \ldots, id^n, time^1, \ldots, time^n, \Psi)$. When, for example, the id attribute of a tuple x has been degraded to attribute state n (notated as $x \in (id^n, time^1)$), then all $x.id^i, i < n$ have the value *null*.

Let U be the set of all user names in state id^1 which all belong to the same group g in a snapshot of a relation R:

$$U = \{x : R \mid x.id^2 = g \bullet x.id^1\}$$

Given an already degraded tuple $x_{tar} \in (id^2, time^1)$ for which we know the user name belongs to U, (hence, $x_{tar}.id^1 = null$ and $x_{tar}.id^2 = g$), $\frac{1}{|U|}$ is the probability P for guessing the original value $x_{tar}.id^1$. Our goal is now to upgrade the *tar*get tuple $x_{tar}.id^1$ to show that privacy can be violated by using available knowledge obtained from snapshot R. Let U' given x_{tar} be the set of names of tuples which where acquired *before* $x_{tar}.time^1$ and are *not* degraded yet:

$$U'(x_{tar}) = \{x : R \mid x.id^1 \neq null \wedge x.time^1 < x_{tar}.time^1 \bullet x.id^1\}$$

Using this set U' of user names which can *not* belong to x_{tar}, the actual probability P' of guessing the original name belonging to the already degraded value x_{tar} is

$$P'(x_{tar}) = \frac{1}{|U| - |U'(x_{tar})|}$$

Indeed, the number of possibilities for the original values is decreased by the knowledge of candidates which could *not* be the original value, and therefor the probability of guessing the original value is increased. Table 1 contains a snapshot with random generated data, with 5 different users all belonging to the same group g. A degraded value is notated with a $-$ followed by the original value surrounded with brackets. At 20:54:56 (row 10) a tuple x_{tar} with original 'name' 4 and retention period $\Psi = 8$ has been acquired and has been degraded at 21.02:56. The 'standard' probability of guessing this value $P = \frac{1}{5}$, but here it is $\frac{1}{4}$ because the set $U'(x_{tar})$ contains 1 values (0). So, there is a privacy violation.

When we make the additional assumption that we can use knowledge about the used policies, than we are able to increase the probability to upgrade a degraded value even more. Let $U''(x_{tar}, t)$ be the set of names of users which can *not* belong to x_{tar} because otherwise x_{tar} would not have been degraded yet given the time $t \leq \{d : U \mid d.name \neq null \bullet min(d.time + \Psi)\}$ the snapshot could have been taken:

$$U''(x_{tar}, t) = \{x : R \mid x.\Psi^1 + x_{tar}.time^1 > t \bullet x.id^1\}$$

Given this set U'', the probability we can guess[2] the original name value of an already degraded tuple x_{tar} and the snapshot time t is defined as:

$$P''(x_{tar}, t) = \frac{1}{|U| - |U''(x_{tar}, t)|}$$

If we look again to the example in table 1, we see that for the tuple which is degraded at 21:02 set $U''(x_{tar}, t) = \{0, 2, 3\}$ (t = 21:03:33). This means that $|U''(x_{tar}, t)| = 3$ and the probability of guessing the original value is $\frac{1}{2}$ instead of $\frac{1}{5}$. Hence, again there is a privacy breach.

	id^1	$time^1$	Ψ	$time^1 + \Psi$	P	P'	P''
0	1	21:00:00	5	21:05:00	1	1	1
1	0	20:59:33	14	21:13:33	1	1	1
2	3	20:59:32	11	21:10:32	1	1	1
3	1	20:59:27	5	21:04:27	1	1	1
4	2	20:59:10	11	21:10:10	1	1	1
5	1	20:59:05	5	21:04:05	1	1	1
6	3	20:58:35	11	21:09:35	1	1	1
7	2	20:57:44	11	21:08:44	1	1	1
8	2	20:56:58	11	21:07:58	1	1	1
9	0	20:55:59	14	21:09:59	1	1	1
10	4	20:55:33	8	21:03:33	1	1	1
11	- (4)	20:54:56	8	21:02:56	$\frac{1}{5}$	$\frac{1}{4}$	$\frac{1}{2}$
12	- (1)	20:54:40	5	20:59:40	$\frac{1}{5}$	$\frac{1}{4}$	$\frac{1}{2}$
13	0	20:54:00	14	21:08:00	1	1	1

[2]In stead of saying "guess the original value" we could better say: "knowing with a probability P that a name belongs to the correct user".

14	0	20:53:20	14	21:07:20	1	1	1
15	- (4)	20:52:59	8	21:00:59	$\frac{1}{5}$	$\frac{1}{4}$	$\frac{1}{2}$
16	0	20:52:19	14	21:06:19	1	1	1
17	- (1)	20:51:47	5	20:56:47	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{4}$
18	- (3)	20:51:27	11	21:02:27	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{4}$
19	- (3)	20:50:40	11	21:01:40	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{4}$

Table 1: *Randomly generated data, with $t \leq$ 21:03:33), 5 different users, uniformly distributed policies with an average Ψ of 10 minutes, and an simulated insert rate of 1 tuple belonging to the group per 2 minutes.*

We could make a union between U' and U'' to try to increase P, which will succeed if $|U' \cup U''| > max(|U'|, |U''|)$, that is $\neg\,(U'' \subseteq U') \wedge \neg\,(U' \subseteq U'')$. Luckily, this situation can never occur. The remainder is devoted to the proof of this fact.

We first look to the case that a certain element x' is in the set of names which could not belong to x_{tar}, because otherwise x_{tar} would not have been degraded yet U'', but is not in the set of names of tuples acquired before x_{tar}, and not have been degraded yet (U'). Hence, to check if U'' can be a subset of U', we check if there are elements in U'' which are not in U':

$$\begin{aligned}\{x : R \mid x.id^1 \neq null \quad &\wedge \quad x.\Psi + x_{tar}.time^1 > t \\ &\wedge \quad \neg\exists\Big(x' : R \mid x'.id^1 = x.id^1 \wedge x'.time^1 < x_{tar}.time^1\Big)\}\end{aligned}$$

In table 1, tuples at row 2 and 4 (with $id^1 = 2, 3$) belong to this set, thus the statement $\neg\,(U'' \subseteq U')$ can be *true* . This situation is likely to occur when retention periods are relatively short compared to insertion rate. The second case however states that an element x' must be acquired before x_{tar} and not have been degraded yet ($x'.id^1 \in U'$), and the retention period belonging to this element x' is so that if it also belongs to x_{tar}, x_{tar} still would have been degraded ($x'.id^1 \notin U''$):

$$\begin{aligned}&\{x : R \mid x.id^1 \neq null \wedge x.time^1 < x_{tar}.time^1 \wedge x_{tar}.time^1 + x.\Psi > t\} \\ =\ &\{x : R \mid x.time + x.\Psi > t \wedge x.time^1 < x_{tar}.time^1 \wedge x_{tar}.time^1 + x.\Psi > t\} \\ =\ &\{x : R \mid t - x.time < x.\Psi < t - x_{tar}.time^1 \wedge x.time^1 < x_{tar}.time^1\} \\ \Rightarrow\ &\{x : R \mid x.time > x_{tar}.time^1 \wedge x.time^1 < x_{tar}.time^1\} = \emptyset\end{aligned}$$

Hence, this set is empty, meaning that $U' \subseteq U''$ always holds, and $\neg\,(U'' \subseteq U') \wedge \neg\,(U' \subseteq U'')$ is *false*. The probability P can not be increased by combining the breaches caused by infering U' and U''. □

5 Metric

5.1 Expected value

In Section 4.1 we defined the upgrade probability as the probability to upgrade *something*.This probability only says something about the probability of upgrading the degraded value, but not about how *much* the refinement is. For example, if data has been degraded to day, you have the highest probability to find the hour, whereas the probability to find the original second is very small. However, only upgrading the hour gives far less information about the actual time than the time in seconds, hence, the privacy breach is less severe.

To visualize this, we make a comparison with the lotteries: the probability of winning a million is much smaller than the probability of winning 10 Euro's, but with by winning the jackpot you get far more rich than by winning only €10. Using the probabilities to win the different prices of the lotteries, we can calculate the *expected amount of money* we win if we buy a ticket. We use the price distribution system of the Dutch 'staatsloterij'; the amount of money you win depends on how many ending digits of your lucky number are correct. Thus, if you have three ending digits correct, you win 1.000 + 100 + 10 = €1.110.

n	*Lucky number*	*Price*	P
4	6789	€10,000	$\frac{1}{10000}$
3	*789	€1,000	$\frac{1}{1000}$
2	**89	€100	$\frac{1}{100}$
1	***9	€10	$\frac{1}{10}$

For example, given above table, the probability of winning exactly €10 is the probability of having exactly only the last digit correct: $\frac{1}{10} \times \frac{9}{10}$. The probability of winning *something* is, analogously to our previous examples in Section 4.1: $\frac{1}{10}$. The *expected value* can be calculated as:

$$E = \frac{1}{10} \times \left(10\right) + \frac{1}{100} \times \begin{pmatrix} 10 \\ + \\ 100 \end{pmatrix} + \frac{1}{1000} \times \begin{pmatrix} 10 \\ + \\ 100 \\ + \\ 1,000 \end{pmatrix} + \frac{1}{10000} \times \begin{pmatrix} 10 \\ + \\ 100 \\ + \\ 1,000 \\ + \\ 10,000 \end{pmatrix} = 4.321$$

What applies for this lottery system also applies for our domain generalization hierarchies: when you know the exact accurate representation of a particular value, you also know the lesser accurate representations. The following table shows the probabilities to guess 'everything' given a particular value degraded to day, up to 'guessing' only the day (and thus upgrading nothing). In this example, we use the amount of money you would 'win' if you upgrade the original value as the *usability function* $f(d^n)$:

	Representation	$f(time^n)$	$P = \frac{1}{\lvert time^n \rvert}$
$\{time^4, \ldots, time^1\}$	*{day,hour,minute,second}*	€10,000	$\frac{1}{86400}$
$\{time^4, \ldots, time^2\}$	*{day,hour,minute}*	€1,000	$\frac{1}{1440}$
$\{time^4, \ldots, time^3\}$	*{day,hour}*	€100	$\frac{1}{24}$
$\{time^4, \ldots, time^4\}$	*{day}*	€10	1

We use the usability function $f(d^n)$ to express the amount of *usability* we expect from *upgrading* the value of attribute d in state n. Using this function we can formulate the *expected usability* $E(d^n)$ given a degraded attribute value in state d^n (hence, d^n itself is visible):

$$\begin{aligned} E(d^n) &= \begin{array}{c} \left(\frac{1}{\lvert d^n \rvert} \times f(d^n)\right) \\ + \ldots + \\ \left(\frac{1}{\lvert d^n \rvert \times \lvert d^{n-1} \rvert \times \ldots \times \lvert d^1 \rvert} \times f(d^n) + \ldots + \frac{1}{\lvert d^n \rvert \times \lvert d^{n-1} \rvert \times \ldots \times \lvert d^1 \rvert} \times f(d^1)\right) \end{array} \\ &= \frac{f(d^n)}{\lvert d^n \rvert} + \ldots + \frac{\sum_{j=1}^{n} f(d^j)}{\prod_{j=1}^{n} \lvert d^j \rvert} \\ &= \sum_{i=1}^{n} \left(\frac{\sum_{j=1}^{i} f(d^{n-j+1})}{\prod_{j=1}^{i} \lvert d^{n-j+1} \rvert} \right) \end{aligned}$$

The difficulty is to define a realistic and practically usefull *usability function* f. For the lotteries we used an amount of money, since this reflects exactly what you get if you win. The amount of usability of data is very user specific and could depend on time (how older the data, the more valuable, or just to other way around). However, we have some options to capture the usability in a general function. First, we can simply use the precision metric as defined by Sweeney[23]. For each individual degraded value of a domain d this precision metric is defined as:

$$f(d^i) = Prec(d^i) = 1 - \frac{i}{n}$$

where n is the attribute state of the first not degraded value d^n. Hence, f is a linear increasing function only taking the degradation step into account.

Another option to define the usability is to take the group sizes into account, or more specific: the number of times the group size has been doubled compared to the group size of the degraded value d^n ($\lvert d^n \rvert$ is according to the definition always 1):

$$f(d^i) = {}^2\log\left(\prod_{i=j}^{n} \lvert d^i) \rvert \right)$$

Analogously to Section 4.1, the group size of the *visible* but *degraded* value d^n is one, thus the usability of the most degraded value (d^m) is in this definition ${}^2\log(1) = 0$. Semantically this means that although an attacker has access to

this value with a probability of 1 (the value *is* visible), it doesn't gain additional usability from it. The same applies for Sweeneys metric: $f(d^n) = 1 - \frac{n}{n} = 0$.

The advantage of the ^{2}log approach is that it is more flexible and takes group sizes into account. Sweeney's metric is linear in the sense that the usability of each step in the generalization hierarchy is decreased with one divided by the length of the generalization hierarchy, even when the provided level of anonymity increases in a non-linear fashion. Using the ^{2}log approach, f is also quite linear (see Figure 11), but not necessarily, depending on the group sizes in the generalization hierarchy which not always exactly double at each generalization step (see Figure 10 in Section 4.1).

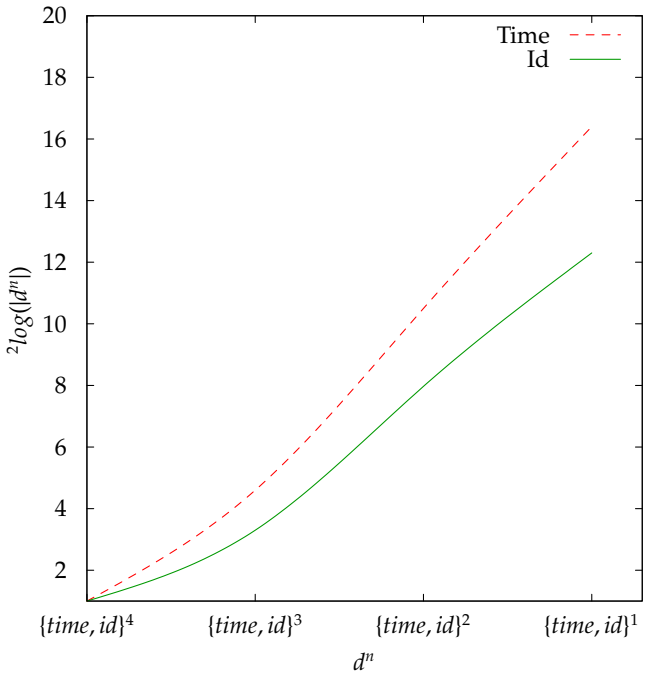


Figure 11: *The usability function using the* ^{2}log *of group sizes is almost linear*

5.1.1 Example of usage

A given tuple has been degraded to state ($time^3, id^2$), representing the time in hours and the id as group member. We first calculate the expected usability

after degrading the time attribute:

$$\begin{aligned} E(time^3) &= \left(\frac{f(time^3)}{|time^3|} + \frac{f(time^3) + f(time^2)}{|time^3||time^2|} + \frac{f(time^3) + f(time^2) + f(time^1)}{|time^3||time^2||time^1|}\right) \\ &= \frac{0}{1} + \left(\frac{0 + {}^2\log|min|}{1 \times |min|} + \frac{0 + {}^2\log|min| + {}^2\log|min||sec|}{1 \times |min||sec|}\right) \\ &= 0 + \left(\frac{{}^2\log 60}{60} + \frac{{}^2\log 60 + {}^2\log 60 \times 60}{60 \times 60}\right) \\ &\approx 0 + \left(\frac{5.91}{60} + \frac{5.91 + 11.8}{3600}\right) \approx 0.103 \end{aligned}$$

and for the id attribute:

$$\begin{aligned} E(id^2) &= \left(\frac{{}^2\log|id^2|}{|id^2|}\right) + \left(\frac{{}^2\log|id^2| + {}^2\log|id^1|}{|id^2||id^1|}\right) \\ &= \frac{{}^2\log 20}{20} \approx \frac{0}{1} + \frac{0 + 4.32}{1 \times 20} \approx 0.216 \end{aligned}$$

So, this examples show that given this metric, the *expected usability* of a time value degraded to the third degradation step is lower than the the expected usability of a id value degraded to the second degradation step. Note however that the usability function used in this example doesn't take the semantics of the domains into account, it only uses the group sizes.

5.2 Privacy metric

In the previous section we extended the notion of *upgrade probability* by applying a usability function to the individual states of an attribute, and use the individual probabilities of upgrading a degraded value to each individual state to calculate the *expected value* $E(d^n)$ an attacker can expect given the degradation hierarchy (and thus given the policy of the user). This expected value expresses an lowerbound of what will be disclosed when a data store is attacked according to the used policies. In this section we use this expected value to create a metric for the amount of privacy one may expect given a certain generalization hierarchy.

The expected value as defined in the previous section can be seen as the *remaining* value of an item after it has been degraded. Hence, if a item has not been degraded yet (it is in state d^1), its value is expressed by the usability function $f(d^1)$. However, analogously to what has been mentioned before: when knowing the accurate value of an item, one also knows the degraded version of this item. Given the length of a domain generalization graph $|dgh|$, we therefor define the *maximum usability* V_{max} as:

$$V_{max} = \sum_{i=1}^{|dgh|} f(d^i)$$

The *total* usability of a degraded item $V(d^n)$ can now be defined as the *remaining usability* of the *degraded* part of that item, plus the usability of the *non-degraded*

part of that item:

$$V(d^n) = \sum_{i=n+1}^{|dgh|} f(d^i) + E(d^n)$$

Given the maximum usability an item had *before* degrading V_{max}, and the usability it has *after* degrading $V(d^n)$, we propose to define the amount of privacy $P(d^n)$ as:

$$\begin{aligned} P(d^n) &= V_{max} - V(d^n) \\ &= \sum_{i=1}^{|dgh|} f(d^i) - \left(\sum_{i=n+1}^{|dgh|} f(d^i) + E(d^n) \right) \\ &= \sum_{i=1}^{n} f(d^i) - E(d^n) \\ &= \sum_{i=1}^{n} f(d^i) - \sum_{i=1}^{n} \left(\frac{\sum_{j=1}^{i} f(d^{n-j+1})}{\prod_{j=1}^{i} |d^{n-j+1}|} \right) \end{aligned}$$

5.2.1 Example of usage

We already calculated the expected (remaining) usability of a tuple which has been degraded to state $(time^3, id^2)$. The maximum usability of a time attribute is:

$$\begin{aligned} V_{max}^{time} &= \sum_{i=1}^{|dgh_{time}|} f(time^i) \\ &= \sum_{i=1}^{|dgh_{time}|} {}^2\log |time^i| \\ &= {}^2\log |sec| + {}^2\log |minutes| + {}^2\log |hours| + {}^2\log |day| \\ &\approx 16,4 \end{aligned}$$

The usability after degradation of $time^3$ is:

$$\begin{aligned} V(time^3) &= \sum_{i=4}^{|dgh_{time}|} f(time^i) + E(time^3) \\ &= f(day) + E(time^3) \\ &= {}^2\log |day| + E(time^3) \approx 0 + 0,103 \end{aligned}$$

Hence, the amount of privacy, expressed as an percentage of the maximum usability V_{max} is:

$$\begin{aligned} \frac{P(time^3)}{V_{max}^{time}} \times 100\% &= \frac{V_{max}^{time} - V(time^3)}{V_{max}^{time}} \times 100\% \\ &\approx \frac{16,4 - 0,103}{16,4} \times 100\% \approx 99,4\% \end{aligned}$$

The amount of privacy of the item in id^2 is much lower:

$$\begin{aligned} V^{id}_{max} &= {}^2\log|name| + {}^2\log|group| + {}^2\log|dept| + {}^2\log|compant| \\ &\approx 12,3 \end{aligned}$$

$$\begin{aligned} V(time^3) &= \sum_{i=3}^{|dgh_{id}|} f(id^i) + E(id^2) \\ &= f(day) + E(time^3) \\ &= {}^2\log|dept| + {}^2\log|company| + E(id^2) \approx 4,64 + 0,22 \end{aligned}$$

$$\begin{aligned} \frac{P(id^2)}{V^{id}_{max}} \times 100\% &= \frac{V^{id}_{max} - V(id^2)}{V^{id}_{max}} \times 100\% \\ &\approx \frac{12,3 - 4,86}{12,3} \times 100\% \approx 60,5\% \end{aligned}$$

Those example shows again the amount of relative privacy one might expect from degrading to different states. Note that given our metric, unless the domain of the attribute is not bound (such as salary for example), the amount of privacy can never be 100%.

5.3 Using the privacy metric

The privacy metric as defined in the previous section can be used to show the effect of the existing inference breaches, and to compare the provided privacy and the required privacy of a set of policies. To do this, the group sizes given the domain generalization hierarchy have to be replaced by the actual group sizes acquired after applying the inference rules as discussed in Section 4.2. In future work, we will experiment with real data sets and policies to measure the amount of privacy we can guarantee by applying our retention model, using this metric.

6 Conclusion

In this technical report we made a beginning with exploring the problems of extending our life cycle policy model with *personalized* policies. The main technical consequence on the storage structure is that the degradation order of tuples is not equal to the insert order anymore when managing multiple policies. This has an impact on the buffered deletion strategy. A preliminary solution to this problem is to buffer inserts, and sort them on degradation order. Besides some technical problems we showed that personalizing the policies brings some inference breaches which have to be further investigated. To make such an investigation possible, we introduced a metric for privacy, which enables the possibility to compare the provided amount of privacy with the amount of privacy required by the policy.

This research is important in order to develop tools to give the donor of data more control over his or her privacy, decreasing the information asymmetry caused by ubiquitous computing. Degradation of data, with degradation

patterns defined by the donor himself, make it clear which data in what form is stored for how long, without having the need to put trust on the service provider infinitely. This report is a first step towards fully personalized life cycle policies.

References

[1] G. Aggarwal, M. Bawa, P.Ganesan, H. Garcia-Molina, K. Kenthapadi, N. Mishra, R. Motwani, U. Srivastava, D. Thomas, J. Widom, and Y. Xu. Vision Paper: Enabling Privacy for the Paranoids. In *Proceedings of the 30th VLDB Conference*, 2004.

[2] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Hippocratic databases. In *28th Int'l Conf. on Very Large Databases (VLDB), Hong Kong*, 2002.

[3] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Implementing p3p using database technology. In *ICDE*, pages 595–606, 2003.

[4] Nicolas Anciaux, Luc Bouganim, H. J. W. van Heerde, Philippe Pucheral, and P. M. G. Apers. Instantdb: Enforcing timely degradation of sensitive data. In *Proceedings of the 24th International Conference on Data Engineering*. IEEE Computer Society Press, April 2008.

[5] Elisa Bertino, Ji-Won Byun, and Ninghui Li. Privacy-preserving database systems. *Lecture Notes in Computer Science*, 3655:178–206, 2005.

[6] Ji-Won Byun and Elisa Bertino. Micro-views, or on how to protect privacy while enhancing data usability: concepts and challenges. *SIGMOD Rec.*, 35(1):9–13, 2006.

[7] Huseyin Cavusoglu, Birendra Mishra, and Srinivasan Raghunathan. The value of intrusion detection systems in information technology security architecture. *Info. Sys. Research*, 16(1):28–46, 2005.

[8] L. Cranor. P3p: making privacy policies more useful. *Security & Privacy Magazine, IEEE Security & Privacy Magazine, IEEE*, 1(6):50–55, 2003.

[9] CSI/FBI. *CSI/FBI computer crime and security survey*. 2005.

[10] Peter Fleischer and Nicole Wong. Taking steps to further improve our privacy practices. The official Google blog, March 2007.

[11] Europese Gemeenschap. Richtlijn 95/46/eg van het europees parlement en de raad. `http://ec.europa.eu/justice_home/fsj/privacy/docs/95-46-ce/dir1995-46_part1_nl.pdf`, 1995.

[12] Rüdiger Grimm and Alexander Rossnagel. Can p3p help to protect privacy worldwide? In *MULTIMEDIA '00: Proceedings of the 2000 ACM workshops on Multimedia*, pages 157–160, New York, NY, USA, 2000. ACM Press.

[13] Hakan Hacigümüş, Balakrishna R. Iyer, Chen Li, and Sharad Mehrotra. Executing SQL over encrypted data in the database service provider model.

In *SIGMOD Conference*, 2002.

[14] Daniel Hillyard and Mark Gauen. Issues around the protection or revelation of personal information. *Knowledge, Technology, and Policy*, 2007.

[15] Xiaodong Jiang, Jason I. Hong, and James A. Landay. Approximate information flows: Socially-based modeling of privacy in ubiquitous computing. In *UbiComp '02: Proceedings of the 4th international conference on Ubiquitous Computing*, pages 176–193, London, UK, 2002. Springer-Verlag.

[16] Marc Langheinrich. A privacy awareness system for ubiquitous computing environments. In *UbiComp '02: Proceedings of the 4th international conference on Ubiquitous Computing*, pages 237–245, London, UK, 2002. Springer-Verlag.

[17] Ashwin Machanavajjhala, Johannes Gehrke, Daniel Kifer, and Muthuramakrishnan Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)*, page 24, Washington, DC, USA, 2006. IEEE Computer Society.

[18] Adam Meyerson and Ryan Williams. On the complexity of optimal k-anonymity. In *PODS '04: Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 223–228, New York, NY, USA, 2004. ACM Press.

[19] Gerome Miklau, Brian Neil Levine, and Patrick Stahlberg. Securing history: Privacy and accountability in database systems. In *CIDR*, pages 387–396, 2007.

[20] Bruce Schneider. Did nsa put a secret backdoor in new encryption standard? http://www.wired.com/, 2007.

[21] Einar Snekkenes. Concepts for personal location privacy policies. In *EC '01: Proceedings of the 3rd ACM conference on Electronic Commerce*, pages 48–57, New York, NY, USA, 2001. ACM.

[22] Patrick Stahlberg, Gerome Miklau, and Brian Neil Levine. Threats to privacy in the forensic analysis of database systems. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 91–102, New York, NY, USA, 2007. ACM Press.

[23] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal on Uncertainty Fuzziness and Knowledge-based Systems*, pages 557–570, 2002.

[24] Google. Google web history. `http://www.google.com/history`.

[25] W3C. Platform for privacy preferences (P3P) project. `http://www.w3.org/P3P/`, June 2005.