Towards an inference detection system against multi-database attacks

Paul Lachat[2,3], Veronika Rehn-Sonigo[1], and Nadia Bennani[2]

[1] University of Bourgogne Franche-Comte, FEMTO-ST Institute, CNRS, France
veronika.sonigo@femto-st.fr
[2] University of Lyon, CNRS, INSA Lyon, LIRIS - France
{ paul.lachat , nadia.bennani }@insa-lyon.fr
[3] University of Passau, Department of Distributed and Multimedia Information Systems - Germany

Abstract. Nowadays, users are permanently prompted to create web accounts when they buy online goods. This collected data gives an insight on the user, sometimes beyond the application scope. Inference attacks on databases represent an issue for data controllers when malicious processors attempt to guess sensitive data - to which they haven't access - by inferring them using legally accessed data. Several inference attack detection systems address this problem in case of a single targeted database. But the issue remains unsolved in case of several databases to which the same users might have submitted their data. In this paper, we propose a global model and its associated graph representation named *Global Instance Graph* (GIG) representing the probabilistic and semantic dependencies inside each database, enriched by the dependencies between the different databases. The graph is obtained using privacy-preserving record linkage techniques and serves as a knowledge input to the inference attack detection system. We validate the GIG creation feasibility thanks to a proof of concept. Despite the quadratic creation time, the performances when data is queried from the databases are not affected since the GIG creation is performed offline.

Keywords: Inference detection · Multi-database attacks · Data privacy · Privacy-preserving record linkage

1 Introduction

We are assisting an era with almost unlimited data storage capacities thanks to the constant increase in data centers' offers. As a consequence, online applications do not hesitate to store huge amounts of raw data concerning their users' habits and behaviors. Moreover, users are permanently prompted to disclose more information including their personal data. But sometimes, this collected data gives an insight on the user beyond the application scope. This could benefit to external stakeholders with the objective to learn more about the user, which could harm the user's privacy. New regulations are now emerging to protect the user privacy, among them the *General Data Protection Regulation* (GDPR)

which is intended to regulate the way EU citizens' personal data should be consumed (according to user's consent) by data controllers and data processors. Moreover the GDPR is not only a theoretic regulation: Ongoing research works on concrete implementations, e.g. [6]. In fact, user's privacy can be disclosed via inference attacks where malicious processors and controllers exploit the possibility to indirectly infer sensitive data to which they haven't access thanks to legally accessed data. This problem can be locally solved using inference systems proposed in the literature [1–3, 7, 10]. However these solutions are not suitable in case of inference attempts where several databases are implied.

Let us consider the following scenario in which a user u registers on two online applications: $Train$ to buy train tickets and $Flight$ for flight tickets. For each application, u gives a set of her personal data to validate the registration process. At this stage, $Train$ and $Flight$ have the ability to protect the user data thanks to: the implemented access control policies and a locally implemented inference attack detection system. Moreover each application is allowed to share with other controllers and processors, with the consent of u, a part of her collected personal data. In this scenario, we could imagine a common processor $Service$ which gather and process travel information coming from several tour operators (e.g., $Flight$ and $Train$). In this case, $Service$ has legal access to subsets of personal data coming from both operators (among them u's personal data subset F_u and T_u from $Flight$ and $Train$ respectively). This creates possible inference opportunities: a part of F_u could be exploited by $Service$ in order to reveal u's sensitive personal data in T_u while bypassing the local inference detection system of $Train$. This scenario highlights the exploitation of inference channels targeting multiple databases through legal access by the same entity. Inference attacks exploiting these inference channels rely on the *distributed dependency strategy* [12]. We will refer to them in the remainder of the paper as *Multiple Database Inference Attack* (MDIA).

Inference attacks happen either in a non-interactive setting (e.g. dump from a database, log files, etc.) or in an interactive setting (e.g. by means of Web browser, localization systems, database queries, and so on). In this paper we focus on the later and thus we give a brief description of the related proposals in the scientific literature. Staddon [10] presents a solution where keys are given to the users in order to generate the required tokens to query the objects in an inference channel and thus prevent its full exploitation. The solution presented by Biskup [1] relies on dynamically adapting policies preventing malicious users to fully exploit inference channels in a logic-oriented information system. The system presented by Brodsky et al. [2] models the functional dependencies of a database to compute the disclosed knowledge each time a query is issued. Then, based on the query log of the issuing user, the system either denies the answer or returns a fake one. Guarnieri et al. [7] propose a system where one module acts as a policy decision point whereas the other checks inference attempts. The latter relies on the security policy defining the inference threshold of the sensitive information and the attacker model describing the user's *a priori* knowledge. Then, based on the inference detection results, the first module decides to deliver

or not the query answer. Chen et al. [3] propose to tackle inference attacks by building first a *Semantic Inference Model* (SIM) representing the probability of attributes to influence others. Then based on the SIM dependencies, a *Semantic Instance Graph* (SIG) reflecting these dependencies at the instances' level in the database is generated and enrolled to detect the inference attacks. All these works present solutions capable of preventing inference attacks on single databases, but they are not adapted for inference attacks when multiple databases are involved.

In this paper we propose an extension of the work of Chen et al. [3] to address the MDIA issue. Our proposal consists in building a *Global Instance Graph* (GIG) by discovering similarities between instances in the SIGs from the concerned databases in order to model inference channels that could be exploited by MDIAs. Moreover, to avoid honest-but-curious behavior, the SIGs are first anonymised. The faced challenge is to discover these similarities on anonymised data. This paper presents the following contributions: (**i**) The GIG creation algorithm based on a set of SIGs, using Bloom Filters [8] to discover instance similarities while preserving the users' data privacy. (**ii**) An architecture that manages the MDIAs for a set of implied databases.

2 Inference detection in case of a single database

In this section we briefly review the approach of Chen which is based on the creation of a *Semantic Inference Model* (SIM). For more details please refer to the original paper. The purpose of the SIM is to represent the inference channels of a database at schema level. It extends the *Probabilistic Relational Model* (PRM) which is based on Bayesian networks. According to [5], the PRM "[...] allows the properties of an object to depend probabilistically both on other properties of that object and on properties of *related* objects". It is made up of two parts: a skeleton and the parameter. The skeleton represents the relations between each attribute and his *parents* which have a direct influence on it. The parameter is the *Conditional Probability Distribution* (CPD) between an attribute and his *parents*. The CPD is computed for a given database and thus represents the distribution of the data at the time of computation. The SIM is composed of three types of links: (i) dependency links: which are the dependencies related to the skeleton of the PRM (ii) schema links: which connect primary keys and foreign keys in the databases tables and (iii) semantic links: which represent dependencies that can be provided by an operator with domain specific knowledge or by analyzing queries issued to the database. The SIM is instantiated with the instances of the database into a *Semantic Instance Graph* (SIG) in order to represent the dependencies at the instance-level. Thus, the nodes in a SIG represent the attribute values of a specific instance in a database. To protect data against inference attacks, sensitive attributes are identified and their inference thresholds assigned. An inference attack is detected once the percentage of confidence about the value of a sensitive attribute exceeds his corresponding threshold. A Bayesian network is instantiated from the SIG for each user in order to keep track of the knowledge she has about the database.

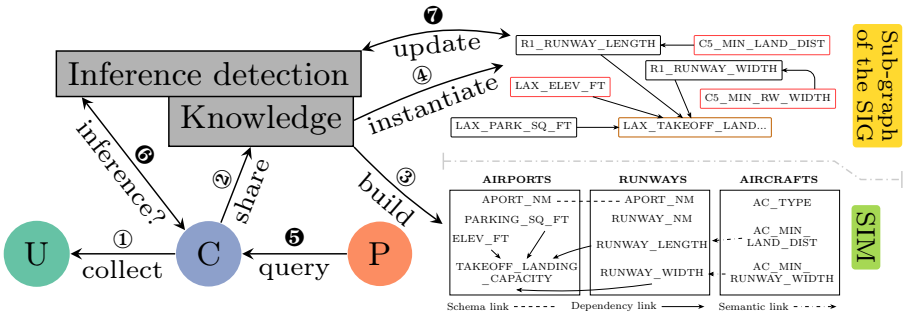


Fig. 1: Workflow of the Chen et al. solution.

The following example, extracted from [3], illustrates the overall workflow of their architecture and the inference attacks detection process. The white steps (i.e., ① to ④) in Figure 1 are made once offline. They represent the SIM and the SIG computing. Whereas the black steps (i.e., ❺ to ❼) are executed whenever a query is submitted to the system. The example represents a chunk of a SIG related to three instances in the database: *LAX*, *R1* and *C5*. For clarity, each attribute instance in the SIG is prefixed with the name of the instance to which it belongs. Thus, multiple occurrences of the attribute RUNWAY_WIDTH could be related to different instances. The example assumes that the attribute TAKEOFF_LANDING_CAPACITY (TLC) is defined as a sensitive attribute and his inference threshold is set to 70%. If a user knows that the instance *C5* is able to land on *R1* from *LAX* and that the instance attributes C5_MIN_LAND_DIST and C5_MIN_RUNWAY_WIDTH have `long` and `wide` values respectively. Then the user can infer the value for TLC with a confidence of 58.3%. If the same user succeeds to obtain the value of the instance attribute LAX_PARKING_SQ_FT (which is `large`), then she can infer the value `'large'` for the instance attribute TLC with a confidence of 71.5% which is above the inference threshold of TLC. Therefore, answering the last query will lead to an inference attack. This solution works well when the queries target the same protected database. However it does not prevent malicious users to query the data required to infer sensitive values both from the protected database and from an external source. This latter scenario is not detected by such a system.

3 Extension to a multi-database protection: A step ahead

Multi Database Inference Attacks (MDIA) occur due to the fact that personal information is scattered among several databases and sensitive data can then be inferred using the knowledge obtained from each database separately. Thus, traditional inference detection systems like [1–3, 7, 10] are not able to catch such attacks since they only model the inference channels present in a single database.

To tackle this issue, our approach proposes to aggregate data controllers' SIGs into one global model called the *Global Instance Graph* (GIG) to be able to represent both inference channels within each database and those implied by the access to several databases. Solving the MDIA issue is not an easy task, we make the following set of hypothesis that remain realistic: (i) We assume that each data controller interested in protecting its users' privacy subscribes to a provider that proposes such a solution and collaborates with it. (ii) The data controllers do not send in clear user data to the inference detection module. (iii) The proposed inference detection module is centralized in order to reuse the inference detection algorithm of Chen by replacing the input SIGs with the GIG. (iv) The inference detection module does not collude with any of the data controllers that subscribe to use its service, but we assume that is could have an honest-but-curious behavior. (v) The databases managed by data controllers are not subject to updates. (vi) The data processors do not collude.

To compute the GIG, one must identify similar instances present in different SIGs. Instances are said to be similar if they represent the same real world entity. For example the instances: (*Alice*, *Thing*, *1992-07-12*) and (*Alice Thing*, *12/7/1992*) have different formats but represent the same real world entity. As a consequence, the GIG must represent those relations of similarity by adding a new kind of links, the *similarity links*, between nodes of similar instances in different SIGs. Adding such links allows to model the propagation of a user knowledge beyond the SIG of the queried database to the other SIGs. In other words, if in the GIG, the nodes n_1 and n_2 from different SIGs are both linked with a *similarity link*, then if a user queries the value of n_1 her knowledge of this attribute value is set to 100% thus the probabilistic propagation will set the percentage of knowledge of the n_2 value to 100%. Therefore, the main challenge of computing the *similarity links* is related to the data format heterogeneity among databases. As demonstrated in [8] *Bloom Filter* (BF) is the most commonly used structure when calculating similarity scores (e.g., based on the *Dice coefficient*). The second challenge to respect hypothesis (ii) is to anonymize data in the SIG before sending it to the inference detection module. To keep the similarity calculation possible, the anonymization function used on each SIG must be the same. The BF must also preserve the privacy of the encoded instances. Encoding techniques such as the one demonstrated in [8] are sensible to attacks, based on frequency accounting or bit pattern, aiming to re-identify data encoded within the BF structure. Such an attack is presented in [4] where the following recommendations to use BF for preserving-privacy computing is proposed: (i) use record-level[4] BF encoding (like the CLK approach proposed in [9]), (ii) employ different hash mechanisms, and, (iii) use advanced techniques (random hashing, adding random bits, etc.).

Computing the GIG Our global graph is initialized by computing the disjoint union of the SIGs issued by the set of data controllers participating to the in-

[4] Record-level mean that each field (i.e., attribute) of an instance are encoded into a single BF whereas field-level mean that each field is encoded into a separated BF.

ference detection solution. Then the GIG computation is completed by linking semantically corresponding nodes related to similar instances from different SIGs with *similarity links*. The naive approach is to first encode instances of each SIG into a BF and proceed to a pairwise similarity score computation. The nodes related to a couple of similar instances are then linked together with a *similarity link* in the GIG. But computing this score for each couple of BFs is expensive. To reduce this cost, we propose to guide the similarity discovery process by beforehand applying schema matching techniques, such as [11], among the SIMs. In fact, a SIM represent dependencies at the schema-level in a database thus by identifying the semantically related attributes in different databases, one can restrict the pairwise similarity score computation to the instances related by a schema matching relation.

Algorithm 1: Computation of the *Global Instance Graph* (GIG)

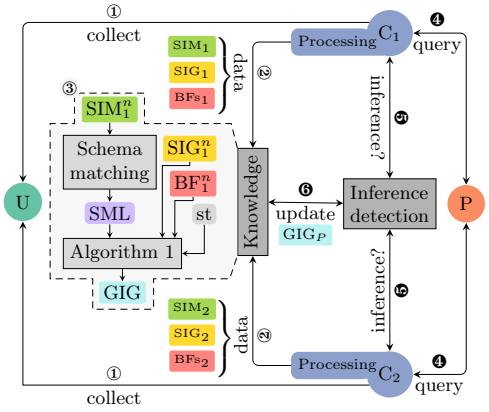
Inputs: $\underline{SIG}$: Set of the data controllers' SIG. $\underline{SML}$: Set of *schema matching relations* between the SIMs. $\underline{BF}$: Set of BFs related to instances in the SIGs. $\underline{st}$: Similarity threshold.

Outputs: $\underline{M}$: Set of pairs of nodes related to semantically similar instances. $\underline{GIG}$: SIGs' linkage based on the instances similarity.

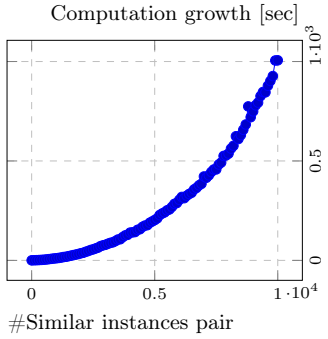
1 **Function** *instance_matching*$(SIG_i,\ SIG_j,\ SML_{ij},\ BF_{ij},\ st)$
2 $M \longleftarrow \emptyset$
3 **foreach** $l \in SML_{ij}$ **do**
4 **foreach** $(n_i,\ n_j)$ *related by* l, $n_i \in SIG_i,\ n_j \in SIG_j$ **do**
`// Get the Bloom Filter of a node's instance`
5 $bf_i,\ bf_j \longleftarrow get(BF_{ij},\ n_i),\ get(BF_{ij},\ n_j)$
6 **if** $Dice\_Coefficient(bf_i,\ bf_j) > st$ **then**
7 $M \longleftarrow M \cup (n_i,\ n_j)$
8 **return** M
9 **Function** *gig_computation(SIG, SML, BF, st)*
10 $GIG \longleftarrow$ disjoint union of the SIGs in SIG
11 **foreach** $(SIG_i,\ SIG_j) \in SIG,\ i < j,\ SML_{ij} \in SML,\ BF_{ij} \in BF$ **do**
12 $M \longleftarrow instance\_matching(SIG_i,\ SIG_j,\ SML_{ij},\ BF_{ij},\ st)$
13 **foreach** $(n_k,\ n_l) \in M$ **do**
14 $GIG \longleftarrow$ Add a *similarity link* between n_k and n_l in GIG
15 **return** GIG

In the following, we present our algorithm for the GIG creation. Algorithm 1 is composed of two functions: (i) *instance_matching* filters pairs of nodes of different SIGs based on the related *schema matching relations* (SML); computes the similarity score of each candidate pair based on *Bloom Filters* (BFs); then collects the pairs of nodes of similar instances based on the similarity threshold, and (ii) *gig_computation* which initialises the GIG with the disjoint union of all the SIGs; processes pairwisely the SIGs to compute the pairs of similar instances; and links nodes related to similar instances with a *similarity link* in the GIG.

The time complexity of Algorithm 1 is related to the number of calls to the $Dice\_Coefficient$ function which computes the similarity score between two BFs (i.e., two instances). In the worst case, for the function *instance_matching* all pairs of nodes $(n_i,\ n_j)$ are related to each semantic matching link l. A similarity score is thus computed for each pair of nodes which leads to a complexity of $O(|SML_{ij}| \cdot |SIG_i| \cdot |SIG_j|)$ where $|.|$ denotes either the cardinality of the set of schema matching relations or the number of instances in a SIG. For readability purposes, the complexity of *instance_matching* is abbreviated as $O(im)$. Then the function *gig_computation* goes through all the combinations of pairs of SIGs without repetition and, in the worst case, calls the function *instance_matching* for each pair. With $|SIG|$ being the number of SIGs, the complexity of *gig_computation* is $O(|SIG|^2 \cdot im)$ since *instance_matching* is called for each combination of SIGs.



(a) Overall workflow of our solution.



(b) GIG computation time growth.

Workflow In our architecture, to achieve the privacy preservation of data controller's information, the values of the attributes in the SIGs and the BFs are anonimized by the data controllers themselves so that there is no need to trust the centralized system. As depicted in Figure 2a, after collecting data from the users ① each data controller must compute its own: SIM, anonymised SIG, and record-level BFs of the instances in the SIG and then sends ② these information to the *knowledge module*. The GIG is built ③ relying on a schema matching technique, such as [11], which processes the set of SIMs from each data controller (i.e., SIM_1, ..., SIM_n denoted by SIM_1^n). Then Algorithm 1 takes as input the computed schema matching relations, the set of SIGs, the related BFs, and the similarity thresholds in order to compute the *similarity links* between the SIGs and therefore create the GIG. Once computed, for each incoming query from a data processor p ❹, the related data controller sends the anonymized answer of the query to the *inference detection module* ❺. Which, as explained in Sec-

tion 2, relies on the *knowledge module* to query the Bayesian network instance of the GIG assigned to the data processor p issuing the query (i.e., GIG_p) ❻. Once GIG_p is retrieved, the *inference detection module* operates the probabilistic propagation for the *dependency links* as in Chen and manages the *similarity links* as presented in Section 3.

4 Experimentation and validation

In this section we have focused on validating and experimenting the GIG creation step as it represents the novel part of our proposal. The project is hosted at: https://gitlab.com/plht/prototype. Since the system proposed by Chen has not been implemented during this experimentation, several programs have been implemented in order to simulate the *processing* block of the data controllers which build the anonymised SIG and record-level BFs as depicted in Figure 2a.

Dataset The requirements that we want for the dataset are: (i) it must contain at least two databases with more than one table to have dependencies between attributes within a table and between different tables (ii) the two schema must have semantically matching attributes (iii) both databases must contain instances that are similar. We did not find a dataset that matches all three requirements. Thus we have adapted the Northix dataset[5] designed for schema matching benchmark in data integration problems. It contains a total of 115 attributes distributed in two databases called *Sakila* and *Northwind*, modeling an online DVD rental store and a fictitious food company respectively. Those databases are often used as samples for learning purposes and experimentation in scientific publications.

The resulting schema matching leads to 110 and 28 relations, between attributes within the same database and between attributes of different databases respectively. In our case, the drawback of this dataset is that it does not contain any similar instances between the two databases. Thus, we have implemented a program which creates either duplicates of customer instances from one database to the other or create pairs of randomly generated similar customer instances in both databases.

Settings & results To measure the efficiency of the GIG computation, we choose to focus only on customer instances matching since they represent the type of instances that interests potentially an attacker in a realistic MDIA. The two SIMs[6] used for the experimentation have been created manually. We have been careful to represent semantically realistic dependencies between the attributes. We have chosen to vary the number of pairs of similar instances from 0 up to 10^4 pairs by inserting 100 new pairs at each step. All points of measure has been repeated 10 times and the median of each repetition is used in the plot. Finally, the measures have been performed in a Docker container hosted

[5] https://archive.ics.uci.edu/ml/datasets/Northix
[6] https://gitlab.com/plht/prototype/-/tree/master/model#experimentation

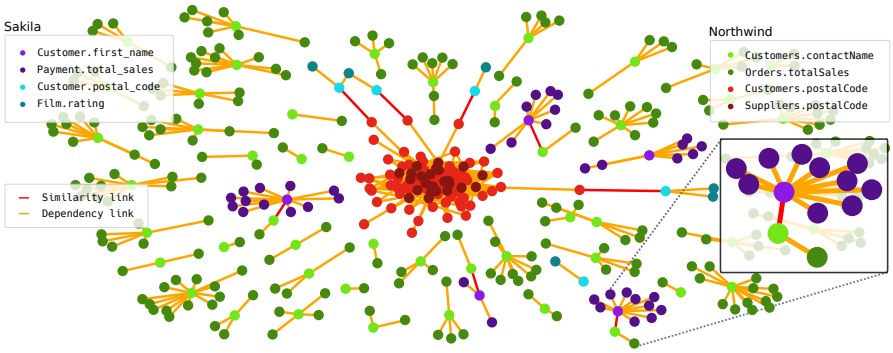


Fig. 3: GIG with four pairs of similar customer instances.

on Ubuntu 14.04.6 and running on an Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz with 46 GiB of RAM.

Figure 2b depicts the measure of the GIG computation time growth depending on the number of pairs of similar instances between the two databases of the Northix dataset. The quadratic growth matches the theoretical time complexity of Section 3 and is the result of processing pairwisely the SIGs (line 11 in Algorithm 1) in order to compute the matching instances. Nevertheless, the GIG can be computed offline by processing the white steps in Figure 2a before being used online and the performances when data is queried from the databases are not affected during the black steps.

Figure 3 shows an example of a small GIG with 4 *similarity links* related to the schema matching relation (*postal_code*, *postalCode*) and 4 others related to (*first_name*, *contactName*). In the zoomed sub-graph the two customer instances are linked together by a similarity link. Therefore, when a query is issued to *Sakila* to get the value of *first_name*, then during the probability propagation phase of the Bayesian network instance specific to the user issuing the query, the inference detection algorithm will update the knowledge of *first_name* to 100% since the user now knows the value of this attribute. Next the *similarity link* will be processed by setting the same percentage of knowledge to the node at the other end (i.e., *contactName*). This knowledge propagation through *similarity links* allows the centralised system to detect knowledge queried on one database which can be used to infer sensitive values on other related databases.

The experimental part of our work highlights several issues linked to the use of BF in our solution: first of all the BF limitations in handling heterogeneity among database schemas to identify similarities[7]. On the other hand, the similarity is stated between two instances based on a fixed threshold.

[7] https://gitlab.com/plht/prototype/-/tree/master/dataset#sakila

5 Conclusion

We have proposed the design of a Multi Database Inference Attack (MDIA) detection system. Our approach extends an existing solution by using schema matching and privacy-preserving record linkage techniques in order to detect inference channels between databases. With respect to our hypothesis, we are able to build a *Global Instance Graph* which represents the inference channels within each (and among) database(s). It allows the detection of MDIA that the usual inference detection systems are not able to identify. This is a first step towards the development of a distributed and fully featured MDIA detection system. In addition, data and schema updates could be integrated in our solution by removing hypothesis (v). In fact, this will affect the models used by the system and requires to propose efficient mechanisms to keep them up-to-date with the concern of maintaining data availability and a good inference detection level.

References

1. Biskup, J.: Dynamic policy adaptation for inference control of queries to a propositional information system. Journal of Computer Security **20**(5), 509–546 (2012)
2. Brodsky, A., Farkas, C., Jajodia, S.: Secure databases: constraints, inference channels, and monitoring disclosures. IEEE Transactions on Knowledge and Data Engineering **12**(6), 900–919 (2000)
3. Chen, Y., Chu, W.W.: Database Security Protection Via Inference Detection. In: Intelligence and Security Informatics. pp. 452–458. Berlin, Heidelberg (2006)
4. Christen, P., Ranbaduge, T., Vatsalan, D., Schnell, R.: Precise and Fast Cryptanalysis for Bloom Filter Based Privacy-Preserving Record Linkage. IEEE Transactions on Knowledge and Data Engineering **31**(11), 2164–2177 (2019)
5. Friedman, N., Getoor, L., Koller, D., Pfeffer, A.: Learning probabilistic relational models. In: Proceedings of the 16th international joint conference on Artificial intelligence - Volume 2. pp. 1300–1307. Stockholm, Sweden (1999)
6. Gerl, A., Bennani, N., Kosch, H., Brunie, L.: LPL, Towards a GDPR-Compliant Privacy Language: Formal Definition and Usage. In: Transactions on Large-Scale Data- and Knowledge-Centered Systems XXXVII, pp. 41–80. Springer (2018)
7. Guarnieri, M., Marinovic, S., Basin, D.: Securing Databases from Probabilistic Inference. In: 2017 IEEE 30th Computer Security Foundations Symposium (CSF). pp. 343–359 (2017)
8. Randall, S.M., Ferrante, A.M., Boyd, J.H., Bauer, J.K., Semmens, J.B.: Privacy-preserving record linkage on large real world datasets. Journal of Biomedical Informatics **50**, 205–212 (2014)
9. Schnell, R., Borgs, C.: Randomized Response and Balanced Bloom Filters for Privacy Preserving Record Linkage. In: 2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW). pp. 218–224 (2016)
10. Staddon, J.: Dynamic inference control. In: Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery (2003)
11. Villányi, B., Martinek, P.: DIPROM: DIstance PROportional Matcher Exploiting Neighbor-levels and Related Terms. Periodica Polytechnica Electrical Engineering and Computer Science **61**(1), 1–11 (2017)
12. Woodall, P., Brereton, P.: A systematic literature review of inference strategies. International Journal of Information and Computer Security **4**(2), 99–117 (2010)