

Formalizing Federated Learning and Differential Privacy for GIS systems in IIIf

Florian Kammüller, Luca Piras, Bob Fields, Raja Nagarajan

Middlesex University London

{f.kammueLLer|l.piras|b.fields|r.nagarajan}@mdx.ac.uk

Abstract. GIS systems, like Google maps or ArcGIS, are an ubiquitous central application but are highly privacy critical. In many GIS systems, inputs from various and diverse sensors potentially expose private information. However, in particular, in the context of public safety, the sensor inputs are crucial to provide timely information for example for weather related warning systems. The notion of Differential Privacy (DP) has become an ad hoc standard, most notably adopted for the US census. Federated Learning (FL) facilitates machine learning for mobile distributed scenarios. This paper proposes a notion of DP for FL for infrastructures with actors and policies formalized in the Isabelle Insider and Infrastructure framework (IIIf). To illustrate this extension of the IIIf by FL and DP on a practical example, we apply the extended framework to a case study from GIS systems for extreme weather warning to control privacy.

1 Introduction

For large distributed systems with various data sources fed by sensors, for example, Geographic Information Systems (GIS) like Google Maps or ArcGIS [7], it is an absolute necessity to be able to use machine learning (ML) in a distributed way on small distributed devices, like IoT devices, and at the same time sharing the data and guaranteeing Privacy according to GDPR (or similar privacy regulations).

The notion of Differential Privacy (DP) has become an ad hoc standard, most notably adopted for the US census [1] to hide personal data in public data records. DP is also used in some GIS systems [7] but so far not much work has been done to explore the practical privacy measure of DP for distributed applications using formal methods for automated verification. Federated Learning (FL) [9] is a widely used algorithm¹ for machine learning in heterogeneous distributed applications. FL facilitates machine learning for mobile distributed scenarios. Its natural relationship to DP has been investigated but the relationship to notions of noninterference in distributed systems is still largely open. Federated Learning and Differential Privacy are naturally related but formally their relationship is concerned with partitioning and distributing data securely over a network

¹ Google scholar counts more than 60K citations for the seminal paper [9].

(the DP part) while safely only returning the stochastic gradient descent (SGD) that ML has procured locally (the FL part). Both concepts have been studied excessively but not many (with a few notable exceptions, for example, [11,8]) have observed that DP resembles the classical notion of noninterference used to express secure information flows. In this paper, we use this observation and extend it even further addressing the distribution and partitioning of the data over a network. We use the expressive power of the Isabelle Insider and Infrastructure framework (IIIf) to support this key issue of Federated Learning. The IIIf supports modeling infrastructures, human actors and policies. It also allows to model data and its distribution across a network as well as the exchange of the SGDs between the client nodes and the server. For modeling the notions of DP, we will reuse the extension of the IIIf with probabilities [6] that supports quantified specification of security and privacy over infrastructures. We can directly express DP as a probabilistic relation over a given infrastructure system. The use of IIIf allows expressing DP in a dynamic way as noninterference while taking into account the nature of FL, that is, the (potentially unbalanced) distribution of data over a network. The novelty of our approach lies in considering DP and FL in the context of dynamically changing (data) systems, that is, with an explicit representation of data distributed on clients and servers as well as representing the federated learning algorithm formally as a protocol. Compared to the vast amount on works exploring DP and FL on applications to data sets, we focus on the dynamicity (of data and communication) and data distribution aspects of DP and FL. Also, by contrast, we formally model all concepts in IIIf thus introducing mathematical rigour and automated verification.

In this paper, we first revisit briefly noninterference and FL (Section 1.1). In Section 2, we present how we express FL in the IIIf. Section 3 presents the central contribution of how DP is used to express noninterference for FL in the extended IIIf with probabilities. This is followed by a result showing how we can use the distributed nature of FL to provide some compositionality property providing a proof technique for proofs of noninterference for FL. To illustrate this extension of the IIIf by FL and DP on a practical example, we apply the extended framework to a case study from GIS systems for extreme weather warning to control privacy (Section 4). The application is motivated by an international research collaboration aiming at upcoming research projects.

1.1 Noninterference and Federated Learning (FL)

Noninterference (NI) [4] is an equivalence property over sequences of state transitions (or sequences of events – depending on the system view). Intuitively, it characterizes that from a certain viewpoint various executions of a system do not reveal any secret information. Classically, that is, for programming languages, noninterference relies on an indistinguishability relation of program executions with I/O or system states. Indistinguishability requires observer viewpoints, usually added simply as H(igh) and L(ow) labels to variables. It then declares system execution (states) to be equivalent if they only differ in values of H-variables. Noninterference can then simply be defined as a Low-bisimulation which means

that in any two program runs the indistinguishability is preserved. This implies that the computation of L-variables is independent of that of H-variables. So, there are no unintended (illicit) information flows from H to L.

Algorithm 1 FL

```

1: Server executes:
2: initialize  $w_0$ ;  $K := \text{set of clients}$ 
3:   for each round  $t = 1, 2, \dots$ 
4:     for each client  $k \in K$  in parallel do
5:        $w_{t+1}^k := \text{ClientUpdate}(k, w_t)$ 
6:        $w_{t+1} := \sum_{k=1}^K \frac{n_k}{n} w_{t+1}^k$ 
7: ClientUpdate( $k, w$ ):
8:    $w := w - \eta \nabla \mathcal{L}(w; P_k)$ 
9:   return  $w$  to Server

```

Federated Learning combines the requirements of training on distributed clients where the data may be unevenly distributed (“unbalanced” [9]) and local data sets may not be representative of the global one (“non-iid” [9]), massively distributed and with limited communication. The FL algorithm can be applied to any finite-sum objective as follows [9]

$$\min_{w \in \mathbb{R}^d} f(w) \quad \text{where} \quad f(w) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^n f_i(w).$$

where the f_i are loss functions, that is, $f_i(w) = \mathcal{L}(x_i, y_i; w)$ where (x_i, y_i) is an example data and w is the model parameter, that is, the current value for the parameters of the ML algorithm to get the best prediction. So, the FL algorithm tries to approximate a prediction function by minimizing a loss function (sometimes called cost function) on data example points (the training data). What is more is that the FL algorithm considers a set of clients K over which the data is partitioned. Let P_k be the partition for client $k \in K$ and $n_k = |P_k|$ the number of data points in that partition. This adds a layer of detail to highlight the unbalanced scenario for the FL algorithm. We can re-write the above description by detailing the averaging in each client.

$$f(w) = \sum_{k=1}^K \frac{n_k}{n} F_k(w) \quad \text{where} \quad F_k(w) = \frac{1}{n_k} \sum_{i \in P_k} f_i w.$$

2 FL in IIIf

To model FL in the IIIf, we represent the FL algorithm introduced in Section 1.1 as a protocol into the IIIf. Since the execution of the FL algorithm is a sequence of exchanges between the server and a number of distributed clients, we use an approach similar to other applications of the IIIf, most prominently the formalization of the Quantum Key Distribution [6], as a protocol built on top of the basic Kripke structures with CTL and Attack Trees.

In general, infrastructures in the IIIf are graphs with actors and local policies attached to the nodes. For the FL algorithm, the actors are the server and its clients attached each to their nodes. The data can be explicitly represented in infrastructures and is also attached to the nodes corresponding to the server and its clients. The constructor `Lgraph` of the data type `igraph` assembles the components `gra`, `aloc`, `server`, `clients`, `ready`, `gradient`, `curmodpar`, `partition`, and `dataset` which represent: the actual topology of locations, the allocation of actors (that is, also devices) to the locations, the server and clients, the clients finished (“ready”) with learning in a round of the FL algorithm, the local gradients currently held in a location, the global current model parameter used for the distributed learning, the partition of the global data set onto the distributed system, and the global data set. The names of these components can be used henceforth as selectors (projection functions) on the `igraph`.

```
datatype igraph = Lgraph gra: location set
                  aloc: actor  $\Rightarrow$  location
                  server: actor
                  clients: actor set
                  ready: actor set
                  gradient: location  $\Rightarrow \alpha :: \text{num}$ 
                  curmodpar:  $\alpha$ 
                  partition: actor  $\Rightarrow$  data set
                  dataset: data set
```

With these infrastructures, we thus can model a distributed scenario. For probabilistic reasoning, we also need the protocol execution traces of the client-server exchanges that form the FL algorithm and attach them as an additional component to infrastructures. The protocol traces are sets of lists of events, where each of these lists represents a finite execution trace of the protocol. For FL these events correspond to the communication actions of the FL algorithm: *putting* the partition of the data on the clients as an initial step, *getting* back the local gradient descents from the clients to the server, and *evaluation* of the combined local gradient descents by the server once all clients have been harvested to compute and redistribute the new current model parameter for the next learning round. The selectors of the following event type are needed for carrying the relevant parameters of the protocol events but their names also convey some intuition about their role in the algorithm, e.g., **datapart** for the data partition.

```
datatype event = Put  datapart: actor  $\Rightarrow$  data set
                | Get  localgrad: client  $\times \alpha$ 
                | Eval newmodpar:  $\alpha$ 
```

We define a datatype for protocol states as sets of lists of events by the single constructor `Protocol` that takes an element of type `event list set` as input. We use the polymorphic Isabelle type constructors `list` and `set` which – applied as postfix constructors to any type β (here `event list set`) – yield the type of sets or lists over β , respectively.

```
datatype protocol = Protocol evs: event list set
```

The actual protocol steps are defined in the inductive definition of a state transition relation that represents the implementation of the FL algorithm. The state over which the transition relation is defined additionally needs to include the distributed system state given by the **igraph** and the policies. Infrastructures are the *states* of the FL system. Infrastructures contain the topology of the network with the current data distribution as well as a current set of protocol traces and the local policies to distribute a global security policy. The policy component **poli** is relevant for applications to define local data and action policies.

```
datatype infrastructure =
  Infrastructure   igra: igraph
                  prot: Protocol
                  poli: [igraph, location]  $\Rightarrow$  policy set
```

We use the underlying theory of Kripke structures and temporal logics CTL in the IIIf to define a specific state transition relation $\rightarrow_{FL}$ that implements the FL algorithm. This relation is an inductive relation over the infrastructure type. It has three cases:

- **put_part** for the initial deployment of the partitioned global data set to all the clients in which the protocol part is presumed to have a freshly started protocol run $[]$ which is then initiated with the **Put [Part]** event where **Part** is a partition of the global dataset of the graph G . This precondition we express using the “disjoint union operator” $\dot{\bigcup}$. The partition **Part** is then inserted into the next state’s graph G' and as a **Put** event into the protocol.
- **get_grad** is the case where a local gradient is computed as the result of the local learning in a client as **Gradc**. The result is loaded into the graphs’ component **gradient** G by updating the position c of this map. Also, the current client c is added to the set **ready** G signifying that c is done now. Note, that we do *not* stipulate $c \notin \text{ready } G$ as a precondition because the FL algorithm allows each client to perform more than one local learning round [9]. A **Get** event is added to an existing protocol trace l .
- **eval_server** finally, pulls together the new model parameter **Newmodel** by averaging all local gradients. These are ready now because the set **ready** contains all clients. After the eval step, **ready** is initialized to the empty set for a new round. This case of the algorithm is noted in a protocol trace by adding the **Eval Newmodel** to an existing trace l .

```
inductive state_transition_FL :: [infrastructure, infrastructure]  $\Rightarrow$  bool
  ("_  $\rightarrow_{FL}$  _")
```

where

```
put_part: G = igra I  $\Rightarrow$  P = prot I  $\Rightarrow$  []  $\in$  evs P  $\Rightarrow$ 
  dataset G =  $\dot{\bigcup}_{c \in \text{clients } G}$  (Part c)  $\Rightarrow$ 
  P' = insertp [Put [Part]] (evs P)  $\Rightarrow$ 
  G' = Lgraph (gra G)(aloc G)(server G)(clients G)
    (gradient G)(curmodpar) Part (dataset G)  $\Rightarrow$ 
  I  $\rightarrow_{FL}$  Infrastructure G' P' (poli I) |
```

```

get_grad:  G = igra I  $\implies$  P = prot I  $\implies$  1  $\in$  evs P  $\implies$ 
  c  $\in$  clients G  $\implies$ 
  Gradc = (curmodpar G) -  $\eta$  *  $\nabla$   $\mathcal{L}$ (curmodpar G, partition c)
  P' = insertp (Get c Gradc # 1) (evs P)  $\implies$ 
  G' = Lgraph (gra G)(alloc G)(server G)(clients G)
        (insert c (ready G))
        (gradient G(c := Gradc))
        (curmodpar G)(partition G)(dataset G)  $\implies$ 
  I  $\rightarrow_{FL}$  Infrastructure G' P' (poli I) |
eval_server: G = igra I  $\implies$  P = prot I  $\implies$  1  $\in$  evs G  $\implies$ 
  ready G = clients G  $\implies$  n = card(dataset G)
  Newmodel =  $\sum_{c \in \text{clients } G}$  (card (partition G c) / n) * gradient G c
  P' = insertp (Eval Newmodel # 1) (evs P)  $\implies$ 
  G' = Lgraph (gra G)(alloc G)(server G)(clients G)
         $\emptyset$ 
        (gradient G)
        Newmodel
        (partition G)(dataset G)  $\implies$ 
  I  $\rightarrow_{FL}$  Infrastructure G' P' (poli I)

```

3 Noninterference for FL using Probabilities and DP

Since the definition of DP is probabilistic, we also reuse the formalization of probabilities that has been provided by the extension of the IIIf for the QKD [6]. In summary, this formalization provides probability reasoning on finite sets of outcomes in the IIIf. It defines probability distributions and probability spaces soundly from first principles, introduces conditional probability and extends the temporal logic CTL of the IIIf to its probabilistic version PCTL. This probabilistic extension also shows the law of total probability useful for reasoning over probabilistic protocols. Although we do not need PCTL in this application, we can use here the theory constructed for it to express DP. In particular, similar to PCTL's J operator [6], we define an operator to express the probability of sequences of state changes. In other word, we define a path quantifier that selects a number of state transition sequences in a Kripke structure and then uses the probability distribution to compute the probability for those paths to occur. This will enable quantifying the likelihood of certain paths that lead to clients transmitting gradient descents to the server. So, the outcomes that are quantified with probabilities by the probability distribution will be paths leading into those “gradient transmission” states. We introduce a “finally” (F) path quantifier. This definition uses two operators from Isabelle's rich theory library on lists: the operator `nth` written as an infix `!` returns the “nth” list element, for example, `[a,b,c] ! 2 = c`; the operator `last` returns the “last”, i.e., rightmost, element of a list, for example, `last [1,2,3] = 3`.

```

definition F :: ( $\alpha$  :: state) set  $\Rightarrow$   $\alpha$  list set
where F s = { 1.  $\forall$  i < length 1. 1 ! i  $\rightarrow_i$  1 ! (Suc i)  $\wedge$  last 1  $\in$  s }

```

We usually want this path quantifier to be relative to a Kripke structure M because we want these “gradient transmission” paths to start from initial states and be paths along states within M . To this end, we define the following specialized version relative to a Kripke model M (the operator hd yields the head (first element) of a list).

definition $\vdash F :: [(\alpha :: \text{state}) \text{ Kripke}, \alpha \text{ set}] \Rightarrow \alpha \text{ list set}$
 $M \vdash F s \equiv \{ l. \text{ set } l \subseteq \text{states } M \wedge \text{hd } l \in \text{init } M \} \cap F s$

Given the operator F to select a set of finite paths leading into a state, we can now combine this with the probability notions provided in the IIIf framework. For example, to introduce the probability for a set of finite paths given an assignment ops of probabilities to paths representing outcomes of protocol runs, the operation pmap ops produces a probability distribution for this protocol. This can then be applied to the previous path selection: $\text{pmap ops } (M \vdash F s)$ computes the probability $P(M \vdash F s)$ of the event that paths end in state s .

Given these notions, we can now express probabilistic noninterference for FL based on DP by stating that for two infrastructure states s_0 and s_1 that both originate from the same initial infrastructure I and have data sets that differ only by one element (expressed using the $\overset{1}{\Diamond}$ operator), we can infer that the probability of finite traces leading into such states that have accumulated the same global model parameter in the FL algorithms as have been accumulated in either s_0 or s_1 differ only by the DP-factor e^ϵ .

definition $\text{NI_FL_DP } I s_0 s_1 e^\epsilon =$
 $I \rightarrow_{FL}^* s_0 \implies I \rightarrow_{FL}^* s_1 \implies$
 $M = \{ \text{Kripke } \{ t. I \rightarrow_i^* t \} \{ I \} \} \implies$
 $G_0 = \text{igra } s_0 \implies G_1 = \text{igra } s_1 \implies$
 $\text{dataset } G_0 \overset{1}{\Diamond} \text{dataset } G_1 \implies$
 $P(M \vdash F \{ s. \text{curmodpar}(\text{igra } s) = \text{curmodpar } G_0 \})$
 $\leq e^\epsilon * P(M \vdash F \{ s. \text{curmodpar}(\text{igra } s) = \text{curmodpar } G_1 \})$

Intuitively, this idea of noninterference corresponds very much to the one of probabilistic noninterference for distributed computations as set out by Volpano and Smith: “under two memories that may differ [only] on H-variables, we want to know that the probability that a concurrent program can reach some global configuration under one of the memories is the same as the probability that it [the concurrent program] reaches an equivalent configuration under the other” [13]. For “concurrent program”, we read “distributed infrastructure”, for “differs on H-variables” we have “the datasets differ only by one” and for “equivalent configuration” we have differential privacy, that is, the probability of reaching equivalent configurations differs only by the DP-factor.

In the literature we see that Goguen and Meseguer introduced unwinding relations to simplify proving that a system has noninterference [5]. As our notion NI_FL_DP of noninterference already has a concurrent nature, it is natural to exploit this for a simplification which is not so much an unwinding but rather a decomposition. Since the FL algorithm exploits the concurrent learning of the

distributed clients, we can break down the global noninterference NI_FL_DP to local DP noninterference for each client. This is given by the following theorem which assumes that for each client that differs from another client in their local dataset by only one item is differentially private in the local learning of the gradients following the FL-algorithm. If this is the case, we can deduce that the global part of the algorithm, that effectively just computes the average of the local gradients, is also differentially private, that is, NI_FL_DP holds.

theorem NI_Decomposition :

$$\begin{aligned}
& I \rightarrow_{FL}^* s_0 \implies I \rightarrow_{FL}^* s_1 \implies \\
& M = \{\text{Kripke } \{t \ . \ I \rightarrow_i^* t\} \{I\} \} \implies \\
& G_0 = \text{igra } s_0 \implies G_1 = \text{igra } s_1 \implies \\
& \text{clients } G_0 = \text{clients } G_1 \implies \\
& (\forall c \in \text{clients } G_0. \text{partition } G_0 \ c \overset{1}{\Diamond} \text{partition } G_1 \ c \longrightarrow \\
& \quad P(M \vdash_F \{ s. \text{gradient } (\text{igra } s) \ c = \text{gk} \}) \\
& \quad \leq e^\epsilon * P(M \vdash_F \{ s. \text{gradient } (\text{igra } s) \ c = \text{gk} \})) \implies \\
& \text{NI_FL_DP } I \ s_0 \ s_1 \ e^\epsilon
\end{aligned}$$

4 GIS case study for extreme weather warning

Climate change causes extreme weather conditions impacting safety and security risks for people and goods. GIS data must be used to provide detailed and multi-faceted geographic and meteorological models, extending existing data sets dynamically with sensor data gathered on the fly by IoT devices. The complexity and dynamicity of these models including processing, storage and analysis is necessary for minimizing the risks and negative impacts, by generating maps for forecasting (e.g., of landslides to alert the population). Such maps need to be simple, accessible and understandable for humans on a much rougher scale than those needed for risk analysis and planning. However, not all policies target easy accessibility and publicity of information. Privacy policies like the GDPR on the contrary request that users may hide information which can also be relevant for GIS data, for example, to protect the competitive advantage of farmers. There are not many studies about privacy techniques applied to agricultural GIS data. A very interesting exception is the work by Nowbakht et al [10] providing a survey on the use of obfuscation methods for protecting privacy of agricultural data to achieve k-Anonymity [10].

In application areas, like location based services, healthcare and criminology the need for privacy might seem natural. In agricultural data, for example data of farms, it may seem exotic. However, an early study on spatial association between diagnosis of Bovine tuberculosis (BTB) and exposure to the parasitic fluke *Fasciola* using 3,026 dairy herds in England and Wales [2] showed that indeed farmland data is highly privacy critical as it might have damaging effects on the competitive advantage.

For extreme weather forecasting, we additionally need real time data gathered by sensors in the field, for example IoT devices running tiny ML for FL. However, the data gathered by sensors in the IoT devices can contain privacy critical detail.

In addition, data distributed by the server to the client devices for training and machine learning may already contain privacy critical data. For example, data on soil classification, so called geoinicators, are crucial for the understanding of the hydrologic conditions of farmland relevant for flood prediction [3].

In order to apply the IIIf framework with the FL-DP extension to this problem domain, we may use a case study implemented as a locale. This allows to instantiate the algorithm and privacy verification methodology to a concrete example. For example, let us consider as a miniature illustration the three client farm locations **Alice**, **Bob** and **Charly** and the server **Sami**. These can then be included in a locale – a feature in Isabelle enabling instances for local reasoning.

```
locale Farms =
  defines Farm_actors = {'Alice'', 'Bob'', 'Charly'', 'Sami''}
```

We can add local assumptions about the example data and the policies to handle it (omitted for brevity). IoT devices with tiny ML are capable of handling policies. The main role of the IIIf here is to break down the global policy into local policies so that these give the right instructions to the IoT to control the collected data and ensure that it does not convey privacy critical details. For DP, the method of choice is to add noise to the data which is done in FL by extracting the learned gradient from the actual data. For example, if an **igraph** element **F** representing a state of the farm network in the locale has a given distribution of the data given by its **partition**, we can assume that another partition of a similar state which differs only by one data element leads to maximally one data element difference in each of the sets **partition F a** for any **a** out of the actors **Alice**, **Bob** and **Charly**. So, for one given global data partition **partition F**, we can compute the least e^ϵ (simply the maximum of all of the clients' e^ϵ s) so that for all $c \in \text{clients F}$,

$$\begin{aligned} & P(M \vdash F \{s. \text{gradient (igra s) } c = gk \}) \\ & \leq e^\epsilon * P(M \vdash F \{s. \text{gradient (igra s) } c = gk \})). \end{aligned}$$

Then the local differential privacy can be guaranteed for this e^ϵ by using the Composition Theorem of Section 3. We arrive at proving global noninterference and thus privacy preservation for the data distribution **partition F** based on the local DP parameters.

5 Conclusions and related work

Nowbakht et al [10] are investigating the effect of obfuscation methods of agricultural data. They compare the use of various obfuscation methods to preserve the k-Anonymity of the data. That is, they try to sufficiently anonymize the data by some kind of blurring of detail to guarantee that the public view of data items never isolates them but leaves k others equal. The k-Anonymity method is highly accepted in healthcare but there is a trend to move to differential privacy (DP), most notably documented in the US adopting the notion for their 2020

census [1]. Another fundamental difference between our works and Nowbakht et al’s [10] is that we provide a formalization of the FL algorithm.

Information theoretic aspects of DP are studied intensively at the moment as is documented in the survey paper [12]. While the foundations appear to become clarified, the relationship to practical implementations of DP, that is, inference or verification systems and thus notions like noninterference are still fairly open. In particular, works that support the formalization and automated reasoning are very scarce with a few notable exceptions. To our knowledge, amongst this there are none in relation to federated learning and modeling distribution of data, communication and processing as we do using the infrastructures, actors, and actions of the IIIf. Tschantz et al [11] already observed the similarity between noninterference and differential privacy with respect to verification. More recently, Liu et al [8] took these concepts further and presented a dedicated model checking technique. However, both works do not consider the conceptual relation to federated learning and thus the distributed aspect of the data and the dynamicity of the system evaluation.

References

1. U. C. Bureau. Disclosure avoidance for the 2020 census: An introduction, 2021. accessed 2.12.2023.
2. J. Claridge, P. Diggle, and C. McCann. Fasciola hepatica is associated with the failure to detect bovine tuberculosis in dairy cattle. *Nat Commun* 3, 853, 2012.
3. S. Fraser and J. Storie. Using geoindicators to prioritize regional wetland locations for flood attenuation in manitoba’s red river basin. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 9:1–10, 08 2016.
4. J. Goguen and J. Meseguer. Security policies and security models. *Symposium on Security and Privacy, SOSP’82*, pages 11–22. IEEE Computer Society Press, 1982.
5. J. A. Goguen and J. Meseguer. Unwinding and inference control. Symposium on Security and Privacy, SOSP’84, pages 75–87. IEEE Computer Society, 1984.
6. F. Kammüller. Attack trees in isabelle extended with probabilities for quantum cryptography. *Computers & Security*, 87, Elsevier, 2019.
7. D. Lavery. The perfect layer effect for differentially-private data: Blur, 2022. accessed 2.12.2023.
8. D. Liu, B. Wang, C. Fu, and L. Zhang. Model checking differentially private properties. *Theor. Comput. Sci.*, 943:153–170, 2023.
9. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. International Conference on Artificial Intelligence and Statistics, AISTATS PMLR54(1273–1282), 2017.
10. P. Nowbakht, L. O’Sullivan, D. P. Wall, and P. Holloway. Implementation of novel polygon-based obfuscation methods to improve privacy of agricultural data. *Trans. GIS*, 27(1):84–104, 2023.
11. M. C. Tschantz, D. K. Kaynar, and A. Datta. Formal verification of differential privacy for interactive systems MFPS, ENTCS 276(61–79), Elsevier, 2011.
12. A. Ünsal and M. Önen. Information-theoretic approaches to differential privacy. *ACM Comput. Surv.*, 56(3), oct 2023.
13. D. M. Volpano and G. Smith. Probabilistic noninterference in a concurrent language. *J. Comput. Secur.*, 7(1), 1999.