



Data Processing Diagrams

A Modeling Technique for Privacy in Complex Data Processing Systems

Job Doesburg¹(✉) , Pascal van Gastel² , Bernard van Gastel¹ , and Erik Poll¹

¹ NOLAI, Radboud University, Nijmegen, The Netherlands

{job.doesburg,bernard.vangastel,erik.poll}@ru.nl

² Avans University of Applied Sciences, Breda, The Netherlands

pptb.vangastel@avans.nl

Abstract. Modern software systems can feature complex data processing, with multiple parties processing various data for different purposes, including training or application of AI. Development of such systems typically involves a multidisciplinary team with different viewpoints. To effectively and efficiently design for privacy requires a multidisciplinary and coordinated effort. We introduce Data Processing Diagrams, an extension of popular Data Flow Diagrams, with standardized notation for fundamental forms of data processing such as data deletion, distribution, encryption and pseudonymization/anonymization. With these extensions, application of well-known privacy design strategies and tactics in complex data processing systems can be reflected. We consider this crucial for unambiguous communication, especially in the earliest design phases of new systems, to quickly compare different architectures and use the models as blueprints for development. We validate the effectiveness of our technique as a shared language between multidisciplinary stakeholders in the context of different co-creation projects that are part of the Dutch National Education Lab AI (NOLAI).

Keywords: Data Flow Diagrams · Privacy Design · Privacy Modeling

1 Introduction

Modern software systems can feature complex data processing, with multiple parties processing various data for different purposes, including training or application of AI. Development of such complex systems typically involves a multidisciplinary team (e.g. development, engineering, legal, governance, ethics, business) with various backgrounds, viewpoints and concerns [9, 10]. Since many privacy features can only be effectively realized by a combination of technical, physical, organizational or legal measures, there can be important interactions between the different disciplines. Currently, however, these stakeholders often work in isolation [14]. This makes designing and communicating about such complex systems challenging: there can be misconceptions, inconsistencies or (at best)

simply inefficiency. Modern (i.e. agile) system development methods, or even modern design methods like participatory design, co-design or co-creation [11], and the intangible nature of software only amplify this. To prevent this requires a simple, standardized, shared language between all stakeholders that can be used from early in the design process and onward.

While Data Flow Diagrams (DFDs) are the de facto standard for modeling data processing systems, they lack standardized notation for specific fundamental forms of data processing, such as data deletion, distribution, encryption¹ (e.g. end-to-end encryption, homomorphic encryption and encryption at rest) and anonymization/pseudonymization. We argue that these concepts are so fundamental to privacy and (consequently) have so many multidisciplinary interactions (i.e. they are relevant not only to technical stakeholders), that standardized notation is required to effectively use these diagrams as a shared language between multidisciplinary stakeholders.

We therefore introduce an extension of DFDs, called Data Processing Diagrams (DPDs), that give a more complete overview of data processing in a system, focussed on privacy, while maintaining a high abstraction level. Specifically, we aim to express well-known privacy design strategies [6] and tactics [3] in our models. We also provide relatively simple (informal) definitions for identifiability, linkability and pseudonymity to characterize data, that are aimed to be understandable by different stakeholders.

Based on the DPD of a system, specific high-level, fundamental system properties can easily be derived, such as which components (or subsystems, or as we will more generally prefer, contexts) could get access to which data (i.e. information flow analysis between systems). As such, a quick privacy assessment of a (proposed) system architecture can be made, comparing different architectures. This can be done completely model-based using only the information in the DPD and without additional background information about specific components. While this quick analysis is in no way an alternative to exhaustive threat modeling using popular frameworks such as LINDDUN [4, 16] or STRIDE [7, 13], we consider this especially useful during the design of new systems when models are used as blueprints and no concrete system details are yet decided.

We consider our models to function as an effective shared language between multidisciplinary (not just technical) stakeholders to unambiguously communicate about the most important features of a complex system's data processing architecture. We validate our modeling technique in the context of NOLAI, the Dutch National Education Lab for Artificial Intelligence, by modeling seven co-creation projects using input from various stakeholders, of which we present a concrete example in Sect. 8.

1.1 Complex Data Processing Systems

We consider our modeling technique to be usable for what we will call *complex data processing systems*, similar to complex systems in other fields of science. The

¹ DFDs typically only consider point-to-point encrypted data flows between two components (encryption in transit), but not encryption across multiple components.

primary goal of complex data processing systems is to process (e.g. collect, store, transform, combine, aggregate, exchange) either personal data about multiple data subjects, or otherwise sensitive non-personal data.

The complexity lies in that various actors (e.g. legal entities) are involved in the system (including, for example, cloud service providers), processing different data for diverse types of functional purposes, in multiple more-or-less autonomous subsystems, possibly with multiple distributed instances of them. For example, data may be used for (1) the basic operation of an application within an organization, (2) centralized training of an AI model, based on data from multiple organizations and (3) validating the effectiveness of the application.

More generally, data processing takes place in a variety of different (types of) contexts, such as legal, physical, organizational, or functional-purpose contexts. Different stakeholders are typically interested in different types of contexts. For all these different contexts, different policies for data protection may be required. As a result, systems may deploy specific Privacy Enhancing Technologies (PETs), including different forms of encryption, pseudonymization or anonymization, resulting in data with different degrees of identifiability, linkability or pseudonymity. This makes analyzing the privacy and security properties of such systems challenging.

1.2 Related Work

Our DPDs extend popular DFDs [5], that, are also used in popular (technical) threat modeling frameworks such as LINDDUN [4, 16] and STRIDE [7, 13]. Because of this, our DPDs can directly serve as input for analysis with these frameworks, only providing more information.

During threat modeling using these frameworks, all DFD components and possible threats to them are systematically identified and assessed. This, however, requires background knowledge about components that may not be available from just the DFD itself. For some types of threats, this can be very fundamental. For example, there is a fundamental difference in privacy risk assessment between a regular data store, and one that stores encrypted data. In a DFD, however, they are displayed as the same component. Our DPDs are able to distinguish these fundamental differences.

We consider this especially useful during early system design, when details about the underlying components are not yet known and the model is used as a blueprint to describe which system architecture should be developed. Our DPDs allow for a fully model-based architectural risk assessment. Comparing different models, instead of threat modeling where the model (i.e. a DFD) only guides the process but cannot replace knowledge of a concrete system.

Some other extensions to DFDs have been proposed to incorporate specific notions of privacy. PA-DFDs [1], for example, extend DFDs with purpose labels to describe the purpose for processing data, aiming to model *purpose limitation*. Sion et al. [14] proposed further DFD extensions for legal concepts and abstractions. While those extensions serve their own specific (analytical) purpose (and could be used in combination with our extensions), they are not able to display

the fundamental forms of data processing required to reflect well-known privacy design strategies (see Sect. 4) in early system design.

2 Background

Our modeling technique extends well-known Data Flow Diagrams (DFDs) [5], consisting of directed graphs of processes, data stores and external entities, with data flows between them. While many grammar rules can be defined for these diagrams [2, 8, 12], we conveniently consider the following informal rules:

External entities are sources or destinations of data and are further not in scope of the system.

Processes do not store data (i.e. for any time longer than the lifetime of the processing itself).

Data stores do not transform data.

Any **data flow** has a process or external entity as either its input or output.

The three main components of a DFD precisely capture all three *states* of data: data at rest (data stores), data in use (processes) and data in transit (data flows). This makes data processing tangible: at the lowest abstraction level, it should be possible to physically pinpoint exactly where each component stores, processes, or transfers data. This results in practical completeness of our models at the lowest abstraction level. Notably, DFDs are also technology-agnostic: they can be used to model both digital data processing by software, but also non-digital or manual processes, which is important for a holistic view of data processing that is required for valid (sound) analysis.

DFDs can be made at different abstraction levels, with the highest abstraction level displaying the whole system as a single process and only displaying the external entities the system interacts with (sometimes called a *context diagram*). Every process (and arguably also every data store) can be unfolded into multiple sub-processes and data stores, with flows between them.² This refinement is crucial for iterative design and communication with different stakeholders, where certain (high-risk) components can be unfolded and collapsed depending on every stakeholder's concerns and background knowledge.

3 Data Properties in DPDs

While only modeling the interaction between different components in a system itself can already be interesting for some stakeholders, this does not exactly describe *what* data is being processed. In order to fully describe this and assess risks, one must describe the exact contents of each data flow as the inputs and outputs of each component. On many abstraction levels, however, this may not be feasible nor desirable. This is especially the case during system design, when only a high-level model is available. Instead, it could be useful to focus on specific properties of data and display these properties using special symbols.

² External entities cannot be refined, as their internals are by definition out of scope.

The exact properties to consider can differ per use case and involved stakeholders, depending on what properties we consider to have inter-stakeholder interactions and be interesting for further analysis, but may often include *identifiability* and *linkability*, especially when applying pseudonymization, anonymization or other PETs in the system, since they have consequences for multiple types of stakeholders. These properties are also fundamental to LINDDUN's main threat types [4]. For more specific use cases, other properties like *repudiability* (if non-repudiation is a desired system property) may also be interesting, or even very domain-specific properties like the nationality of data subjects³. Apart from that, it can be useful to describe data with a general category (such as *medical*, *financial* or generally *sensitive*). As a first step in the modeling process, the relevant data properties between all stakeholders should be established.

3.1 Identifiability, Linkability and Pseudonyms

We specifically propose some practical, relatively simple definitions for **identifiability** and **linkability** (with corresponding graphical representations), because of their interplay and since they are fundamental to privacy and affect almost all viewpoints in some way.

It is hard to precisely define the identifiability of data. While some data may be non-identifiable to most, it could be identifiable to others that have the ability to link data with other identifiable data, making the data indirectly identifiable ultimately. The concepts of identifiability and linkability are thus closely related.

We also discuss **pseudonyms** as a special form of linkability and as a simple yet powerful PET without irreversible loss of data utility. By properly using pseudonymization, namely, one can enforce that different pseudonymous (sub)sets of data cannot be recombined even if one or more parties or components in the system are compromised.

Existing (e.g. mathematical) definitions for these concepts are more formal and strict, but are consequently difficult to understand for many (non-technical) stakeholders and less suited to characterize data flows in a larger system. We thus seek for a compromise between expressiveness, correctness and simplicity.

Definition 1 (Identifiability). *The extent to which data identifies a natural person. We distinguish:*

- ⊕ **directly identifiable:** data identifies a natural person directly, based on publicly available knowledge
- ⊕ **indirectly identifiable:** data identifies a natural person, but requires some non-public knowledge outside the system
- ⊗ **de-identified:** data does not contain any identifiers⁴
- ⊗ **non-personal:** data does not relate to persons at all (Fig. 1)

³ For example, when dealing with various national data processing regulations.

⁴ Some may falsely refer to this as ‘anonymous’ data. Truly anonymous data is additionally unlinkable, or even has any distinguishable statistical features removed.

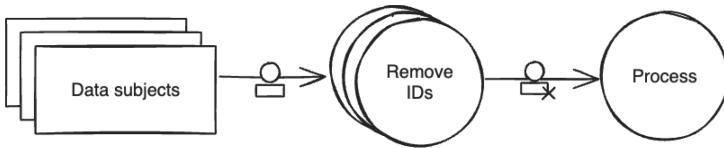


Fig. 1. Example of identifiability annotations.

Definition 2 (Linkability). *The extent to which data can be linked to other data based on an equality or known correlation. We distinguish:*

- ∅* **universally linkable:** high-entropy data that is considered to have universally unique attributes (e.g. biometrics, MAC address)
- ∅* **locally linkable:** limited uniqueness only within the local context (e.g. session IDs or data with random noise)
- ∅* **unlinkable:** non-unique data that is thus inherently unlinkable

Definition 3 (Pseudonymity). *A special form of local linkability without direct identifiability, where local linkability is introduced through a dedicated attribute. Depending on whether data is also indirectly identifiable, we distinguish:*

- ∅* **strict pseudonymous:** data is pseudonymous but further de-identified
- ∅* **soft pseudonymous:** data is both pseudonymous and indirectly identifiable (based on other knowledge than the pseudonym)

Under these definitions, **identifiability** (either direct or indirect) **implies universal linkability**, but **linkability does not imply identifiability**. De-identified but linkable data, this way, can become (either directly or indirectly) identifiable after linking with other (directly or indirectly) identifiable data.

All definitions are relative to some extent and require assumptions on other publicly available data outside the system. For example, DNA samples are obviously unique and would identify a data subject in that sense. However, if we do not assume DNA samples to be publicly available information (which is generally the case), we will not classify a DNA sample as directly identifiable, but rather universally linkable and indirectly identifiable, or even de-identified if we consider DNA samples to not be available to any party at all.

The threshold for all definitions can be subjective, too. In case the combination of postal code and birth year does not relate to a single natural person, but to two or three, one may or may not consider this data identifiable, depending on the use case. During modeling, stakeholders should agree on reasonable assumptions on these matters.

3.2 Implicit Data

When characterizing data flows, it is crucial to also consider *implicit* (meta)data, especially when talking about identifiability and linkability. A person's first name, for example, can hardly be considered identifiable in a global data set, but within a data set of employees of a specific department it most definitely can be! In this case, the department of employment is implicit data, that should be derived from the local context of the processing. Similarly, any data originating from a specific known personal device can be considered linkable to that device and perhaps identifiable. A specific example is real-time data, where the timestamp of interaction could be considered a linkable or even identifiable, implicit attribute, for which we propose special annotation (\mathfrak{D}).

4 Privacy Design Strategies

In 2014, Hoepman published an overview of eight privacy design strategies [6], with various underlying privacy design tactics [3], based on a large literature study. They are considered to form a complete overview of (high-level) approaches to privacy-friendly data processing that can categorize PETs. Specifically, Hoepman distinguishes four data-related and four process-related strategies, with the first category focussing on the actual system architecture and the latter category on more peripheral procedures for governance and compliance. In this section, we show the DFD extensions required to distinguish the four data-related strategies and underlying tactics.

From the in total four data-related strategies, Hoepman identifies two strategies (SEPARATE and HIDE) to limit the *chance* of "privacy violations" in a system, and two (MINIMIZE and ABSTRACT) that limit the *impact* of such violations [6]. Orthogonally, two strategies (HIDE and ABSTRACT) affect the actual data being processed itself, and two strategies (SEPARATE and MINIMIZE) that affect the way the processing of data is organized.

	Change infrastructure	Change data
Reduce impact	MINIMIZE	ABSTRACT
Reduce chance	SEPARATE	HIDE

MINIMIZE. The MINIMIZE strategy simply entails to process only the data that is strictly necessary for the purpose. This can mostly (for the SELECT, EXCLUDE and STRIP tactics) be expressed in our models by annotating the data flows between components with data properties, describing what kind of data is being processed and resulting in fewer data being processed.

For the DESTROY tactic, deleting data from a data store, however, we introduce a new (complemental) type of data flow, that deletes data from a data

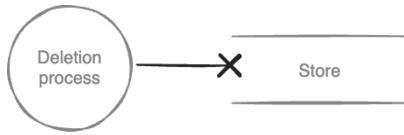


Fig. 2. Notation for data deletion.

store (see Fig. 2)⁵. As a result of making data deletion explicit, it is possible to highlight components where data is stored but never deleted, possibly resulting in orphaned data.

ABSTRACT. The ABSTRACT strategy, consisting of the SUMMARIZE and GROUP tactic, can also be expressed in our models by annotating the data flows with data properties, describing exactly what kind of data is being processed, with data being transformed to be less sensitive, identifiable or linkable.

SEPARATE. The SEPARATE strategy aims to not process (or store) data at a single place, but split it over different components, thus reducing the impact in case of compromise (“violation”) of one of them. This concerns two tactics, DISTRIBUTE and ISOLATE. Isolation is reflected by having multiple different types of components processing different types of data, which can easily be reflected in our models. Distribution, on the other hand, considers the usage of multiple instances of the same type of (sub)system or component for different data subjects. The existence of such components is also typically a characteristic of complex data processing systems, as defined in Subsect. 1.1.

In a classical DFD, every store, process or external entity typically represents a different *type* of component. Any distribution of instances is left implicit or must be specified in natural text, which could leave room for ambiguities. We therefore introduce notation to display multiple instances of the same component type that are separated (Fig. 3) to a certain extent. Related to the distributed components, refinement (unfolding) might result in a subset of components that together functions as an independent subsystem.

For each distributed component or subsystem, the variable or level of distribution should be specified. This could, for example, be the data subject-level or organization-level, with one instance of the subsystem for every data subject or organization respectively. This distribution can also be nested.

HIDE. Finally, the HIDE strategy consists of the RESTRICT, MIX, DISSOCIATE and OBFUSCATE tactics. On an architectural level, the RESTRICT tactic considers employing an access control process, which as such can directly be reflected in our model, but does not influence other architectural features of data processing in a system. Similarly, the MIX and DISSOCIATE tactics are also displayed by specific centralized processes resulting in data with limited linkability or identifiability,

⁵ Data deletion flows *only* work towards a data store!

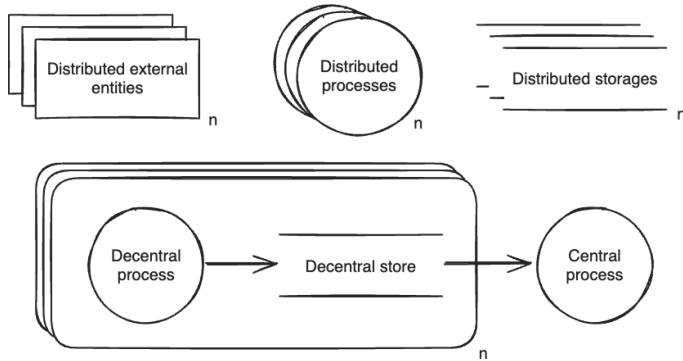


Fig. 3. Notation for (n) distributed components.

or eliminating (real-time) linkable data flows. Cryptographic technologies such as differential privacy also fall under these tactics.

The **OBFUSCATE** tactic, however, is observed in cryptographic processes, encrypting the data and hiding it across part of the processing, towards those unable to decrypt it.

In regular DFDs, data flows are point-to-point between two components (stores, processes or external entities). When encryption is applied, this encryption is typically considered to be point-to-point too, and not end-to-end (across multiple components).

When data is end-to-end encrypted while flowing through multiple components, there are different ways of modeling this. Either, the DFD only describes the high-level data flow between the two endpoints and ignores the existence of all underlying infrastructure, or, more commonly, we ignore the fact that end-to-end encryption is applied and only keep it in the back of our mind when using the DFD for further analysis (perhaps by including some ad-hoc annotations). Finally, one could include explicit encryption, decryption, and key management processes in the model. This, however, would require a lower abstraction level and making the model more complex to analyze.

Therefore, we introduce abbreviated notation for **end-to-end** encrypted data flows across multiple components (see Fig. 4). Across a data store, this is known as ‘encryption at rest’. Across a process, transforming the data while being encrypted, this is known as ‘homomorphic encryption’. Finally, we also introduce notation for multiparty computations (MPC) as a distributed process on encrypted data.

Notice that in the abbreviated notation, key management is ignored, since we consider this a technical detail that is not relevant at higher abstraction levels. For MPC, notice the slightly different notation than for (normal) simple distributed components (see Fig. 3), since for MPC the individual nodes are not fully isolated from each other and are not autonomous. Annotations should

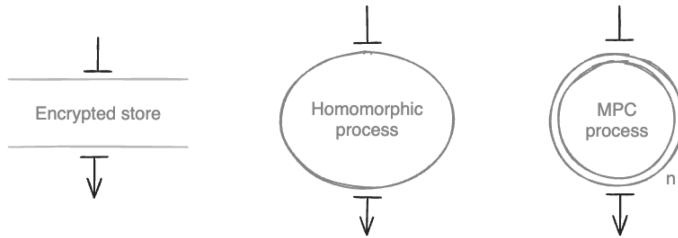


Fig. 4. Abbreviated notation for cryptographic components.

display to what extent the different nodes of an MPC process are separated, or alternatively, MPC processes can be displayed in more detail as distributed processes.

5 Context Boundaries

The STRIDE security threat modeling framework [7] introduced the concept of *trust boundaries* to Data Flow Diagrams, to indicate where in a system trust levels are changing and, for example, a different threat model must be considered. This typically considers the location of a component in the system, either physical or logical (as part of a larger system, network, etc.). The LINDDUN privacy threat modeling framework [16] also briefly mentions trust boundaries, without really describing a specific meaning to them, but similar boundaries could be considered (e.g. legal boundaries).

In complex data processing systems, as described in Subsect. 1.1, typically many (and many types) of these boundaries exist, which we referred to as *contexts*. We therefore generalize the term *trust boundaries* to *context boundaries*, where different contexts might be identified by different characteristics, including but not limited to technical or organizational trust. Exactly which context boundaries to consider can differ per use case, similar to which data properties one should consider as discussed in Sect. 3. Examples could include physical locations, technical or logical separated units (such as different system users, devices, or networks), legal or organizational responsible entities, or natural persons being actors in or having access to components.

By specifying context boundaries, different (types of) contexts are established, resulting in two ways of visualizing them (see Fig. 5). These different contexts could be considered as separate overlays on a single base model. Multiple distributed instances of contexts can also exist.

It is important to notice that with refinement of diagrams, unfolding or collapsing (groups of) components, it is possible that at higher abstraction levels context boundaries cannot always be exactly drawn between distinct components. This results in rather blurry context boundaries somewhere inside a store or process, or components being in multiple contexts (of the same type). For

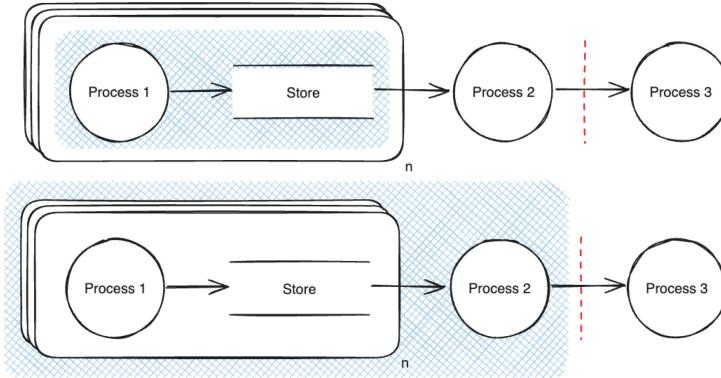


Fig. 5. Different ways to display contexts (blue) or context boundaries (red). (Color figure online)

some types of contexts, it is also possible that components are fully part of multiple contexts at any abstraction level (such as processing purpose or legal entity responsible, in case of shared responsibility between multiple parties). We consider this a feature of our models, indicating that more refinement is required, or that a component requires extra attention.

5.1 Alignment

As already originally mentioned for STRIDE, where data crosses a trust boundary, certain mitigating measures (e.g. technical, legal) may be required. Additionally, we argue that it is exactly where different types of contexts do **not** align with each other, where extra attention is required during system design.

As an example (Fig. 6), the paper archives (data store) from organization A (legal) may be located in the building of organization B (physical). The physical and legal context boundaries do not align. It may thus be possible for organization B to physically get access to the stored data from organization A before process 2 (perhaps, anonymization) has been performed, which perhaps should be resolved by contractual agreements forbidding it, or physical or technical measures, such as placing it in a vault or applying encryption. Here, the interplay with encrypted data flows as discussed in Sect. 4 becomes apparent, since an end-to-end encrypted data flow effectively may not pass a context boundary while a regular flow does.

Inspecting components where context boundaries do not align or are not clearly defined, especially when processing sensitive or identifiable information, allows for prioritized (and thus, more efficient) refinement of a DFD to lower abstraction levels during system design.

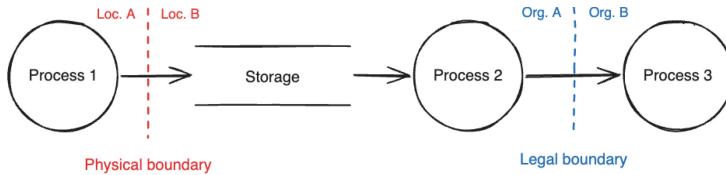


Fig. 6. Two types of context boundaries that are not aligned.

6 System Properties

Based on our DPDs, specific system properties can be derived, using basic graph theory. This can be useful to make specific (privacy) claims about forms of data processing that does **not**⁶ (or *cannot*, because data is not linkable) take place in the system. For example, one could make statements about:

Which components or contexts process which types of data (i.e. where data flows from one context to another)

The existence of data stores where data is not being deleted

To which degree specific types of data are processed centrally or decentrally
Components that do not align with different context boundaries

Whether different data could be combined within a context, based on linkability of the data

Special attention could go out to system properties assuming compromise of one or more contexts, making sure that specific properties still hold after compromise by considering the transitive closure of the graph.

Based on this analysis, high-risk components could be identified, which could then be further refined, until so many details are described that the residual risk either can be accepted, or the risk is mitigated in another way. Ongoing research by the authors focuses on more formal methods for analysis, risk assessment and model refinement during system design using these data processing diagrams.

7 Evaluation

Our modeling technique is being used in the Dutch National Education Lab AI (NOLAI). At NOLAI, every year, about 10 new three-year co-creation projects start (€3M per year), aiming to combine academic research towards AI in education with software product development. Currently, 17 projects are running, and another 80 are expected for the coming years. In these projects different stakeholders from education, industry, and academia are involved. Many

⁶ We can only make *negative* claims, since components might be unavailable, potentially blocking any data processing.

projects feature a scientific experiment that builds on existing platforms, making the responsibilities and risks hard to manage. As such, these projects are typical examples of complex data processing systems. To organize data processing responsibly in all these different contexts, various measures are implemented, including pseudonymization and different forms of encryption, both in transit and at rest.

The authors have modeled eight of these co-creation projects, mostly in the early stages of the development process. This was done in collaboration with various stakeholders, including researchers (in different academic fields), co-creation managers, legal experts, software engineers and ethicists.

While not all stakeholders, especially not those without a technical background, were able to actively create these models from scratch, all were able to understand the models when creating them together with a more technical expert. Moreover, after working with them for a while and with help of templates and examples, most were able to create and interpret simple models independently. Meanwhile, for the more technical stakeholders, the models were unambiguous and contained enough information to perform some high-level risk analysis and compare different designs based on their privacy properties.

More importantly, authors have experienced that the process of creating a DPD required stakeholders to critically think about data processing and define scopes. Making the models identified unclarities or ambiguities and forced stakeholders to address those, early in the process. More than once, the modeling process resulted in different versions of the system between which a choice had to be made, of which many stakeholders were previously unaware. As such, the models functioned as a starting point for discussion between stakeholders, and effectively served as a shared language.

Anecdotally, for several projects, incompatibilities were discovered during modeling, for example where processed anonymized data had to be returned to the original data subjects, or where no procedures were designed to transfer data between two stores (which, in practice, resulted in people using insecure email to transfer sensitive data). The simple process of creating these models, highlighted these blind spots.

In future research, authors plan to further incorporate the modeling technique into a structured design methodology, and validate the effectiveness of the methodology over a period of multiple years of development of a system.

8 Example

As a concrete example, we present NOLAI's VIAT (Video Interaction Analysis Tool) project. In the VIAT project, software is developed to analyze in-classroom video footage for teacher's training purposes, such as detecting moments that a teacher answers or ignores a student's question. Data processing includes the recording, storage and playback of videos, as well as analysis (creating reports) using AI models. Additionally, to develop the tool, selected recorded videos are annotated and used to train these models. VIAT is tested and developed in pilots

at primary schools, and the effectiveness is studied using interviews and analysis of additional data sources by researchers from NOLAI, potentially in multiple independent studies.

Different (legal) organizations are involved in the processing, including several schools, commercial parties (for both the recording platform and AI analysis), cloud service providers and a research institute. As such, data processing takes place in different contexts. In this example, we will focus on legal contexts.

There are several desired privacy features for these contexts. For example, scientific data analysis may only use pseudonymous data. Also, AI model training should not receive identifiable data. Preferably, recorded videos are not accessible to anyone except the school. Finally, the commercial party may not receive the data collected in the scientific context.

In Fig. 7, we present two DPDs for two possible data processing architectures for the VIAT project, where one has obviously stronger privacy guarantees than the other (see captions). Notably, the DFDs for both architectures would be **the same**, illustrating the added value of DPDs over DFDs.

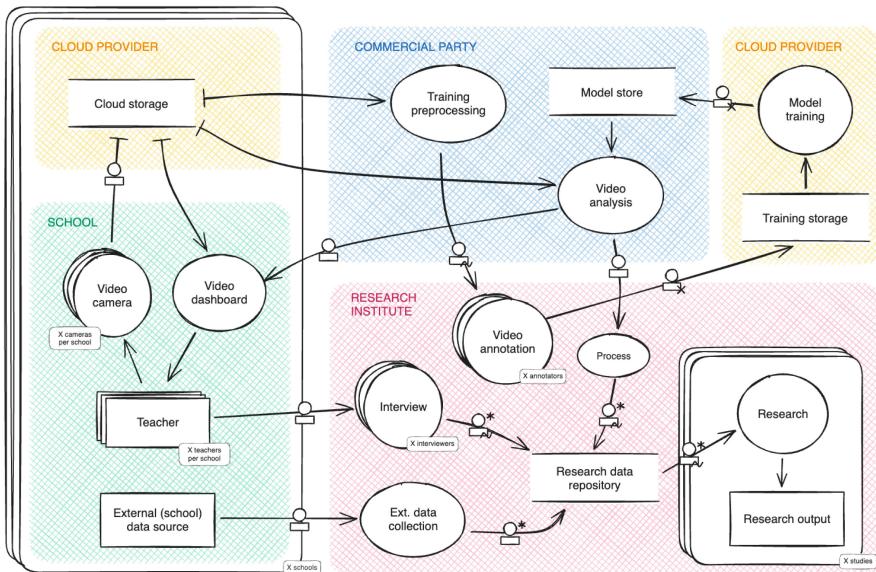
9 Future Work

As presented in Sect. 6, our diagrams can be used to systematically derive system properties based on annotations of data types and assumptions on underlying components as in Sect. 3. To derive more specific properties, more strict mathematical definitions such as k -anonymity [15] could be considered and algebraic rules for such properties, based on these diagrams, could be built.

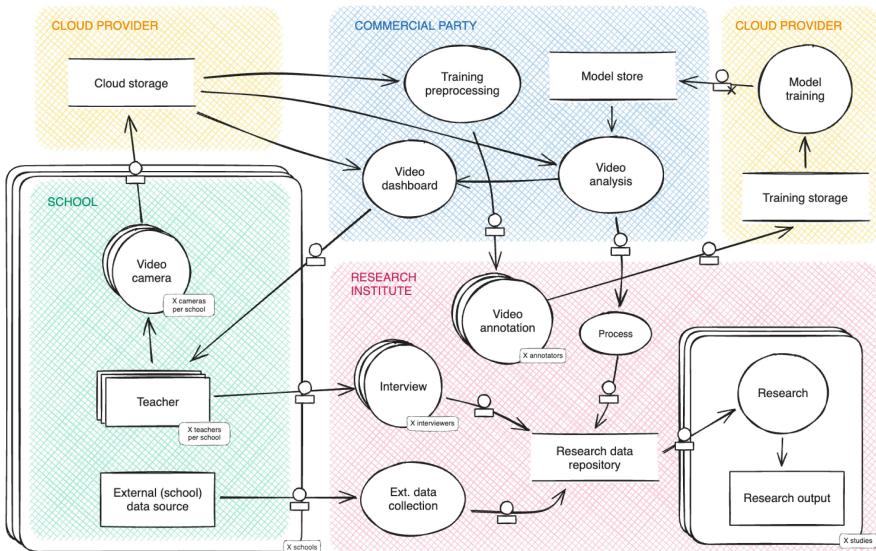
When using data processing diagrams during system *design*, a structured formal method for risk assessment can also be considered, systematically identifying high-risk components and refining them or mitigating risks. While we did show the completeness of our modeling technique with respect to well-known privacy design strategies and tactics [3], choosing which tactic to apply while designing a system, requires thought-out decisions. Future research can focus on designing systematic methodologies to apply specific tactics and identify common privacy design patterns and antipatterns, implementing specific measures.

A different application of our diagrams could be in the analysis of energy consumption of (software) systems, which is, next to privacy and security, also an important factor in implementing data processing sustainably.

Finally, it would be interesting to see to what extent Data Processing Diagrams, at a high abstraction level, can be used as a communication tool for non-expert users (or data subjects) to offer transparency about data processing.



(a) A strong privacy architecture, with mostly **decentralized** data processing at schools (except video analysis), **encrypted** cloud storage **per school**, model training on **de-identified** data, and **pseudonymization** for research.



(b) A weak with fully **centralized** data processing by the commercial party and cloud provider, with **no encrypted storage** at the cloud provider, involving model training on **identifiable** data and lacking **pseudonymization**.

Fig. 7. DPDs of two possible architectures for the VIAT project, with data identifiability annotations.

10 Conclusion

We introduced Data Processing Diagrams, a modeling technique for privacy in complex data processing systems, based on well-known Data Flow Diagrams. The extensions for data deletion, distribution, encryption, together with annotations of identifiability, linkability and pseudonymity of data, are able to reflect application of well-known privacy design strategies [6] and tactics [3]. The resulting models allow for unambiguous communication between multidisciplinary stakeholders (e.g. security, legal, management, ethics, development and engineers) about privacy and data processing in a system, at different abstraction levels. Moreover, specific system properties can be derived from these models. Authors have experienced the models to be effective tools in the system development process, helping to derive system properties and perform risk assessment or threat modeling. In future research, our modeling technique can be further formalized to derive more specific system properties in a systematic way, based on more formal definitions, and perform risk assessment to effectively implement specific design patterns for improved privacy.

Acknowledgments. This research is performed in context of the Dutch National Education Lab AI (NOLAI), funded by the Dutch National Growth Fund.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Alshareef, H., Tuma, K., Stucki, S., Schneider, G., Scandariato, R.: Precise analysis of purpose limitation in data flow diagrams. In: Proceedings of the 17th International Conference on Availability, Reliability and Security, ARES 2022. ACM (2022). <https://doi.org/10.1145/3538969.3539010>
2. Ambler, S.W.: Data Flow Diagrams (DFDs) (2006). Personal webpage. <http://www.agilemodeling.com/artifacts/dataFlowDiagram.htm>
3. Colesky, M., Hoepman, J.H., Hillen, C.: A critical analysis of privacy design strategies. In: 2016 IEEE Security and Privacy Workshops (SPW), pp. 33–40. IEEE (2016). <https://doi.org/10.1109/SPW.2016.23>
4. Deng, M., Wuyts, K., Scandariato, R., Preneel, B., Joosen, W.: A privacy threat analysis framework: supporting the elicitation and fulfillment of privacy requirements. Requir. Eng. **16**(1), 3–32 (2011). <https://doi.org/10.1007/s00766-010-0115-7>
5. Yourdon, E., Constantine, L.L.: Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design, 2 edn. YOURDON Press (1975)
6. Hoepman, J.-H.: Privacy design strategies. In: Cuppens-Boulahia, N., Cuppens, F., Jajodia, S., Abou El Kalam, A., Sans, T. (eds.) SEC 2014. IAICT, vol. 428, pp. 446–459. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55415-5_38
7. Howard, M., Lipner, S.: The Security Development Lifecycle. Microsoft Press (2006)

8. Kozar, K.A.: The technique of data flow diagramming. University of Colorado (1997). Personal webpage <https://spot.colorado.edu/~kozar/DFDtechnique.html>
9. Maier, M.W., Emery, D., Hilliard, R.: 5.4.3 ANSI/IEEE 1471 and systems engineering. INCOSE Int. Symp. **12**(1), 798–805 (2002). <https://doi.org/10.1002/j.2334-5837.2002.tb02541.x>
10. Rozanski, N., Woods, E.: Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. AWPC (2005)
11. Sanders, E., Stappers, P.J.: Co-creation and the new landscapes of design. CoDesign **4**(1), 5–18 (2008). <https://doi.org/10.1080/15710880701875068>
12. Sauter, V.: Data Flow Diagrams. University of Missouri, St. Louis (2002). Personal webpage. https://www.umsl.edu/~sauterv/analysis/dfddfd_intro.html
13. Shostack, A.: Threat Modeling: Designing for Security. Wiley (2014)
14. Sion, L., et al.: An architectural view for data protection by design. In: 2019 IEEE International Conference on Software Architecture (ICSA), pp. 11–20. IEEE (2019). <https://doi.org/10.1109/ICSA.2019.00010>
15. Sweeney, L.: k-anonymity: a model for protecting privacy. Int. J. Uncertain. Fuzziness Knowl. Based Syst. **10**(5), 557–570 (2002). <https://doi.org/10.1142/S0218488502001648>
16. Wuyts, K., Scandariato, R., Joosen, W.: Empirical evaluation of a privacy-focused threat modeling methodology. J. Syst. Softw. **96**, 122–138 (2014). <https://doi.org/10.1016/j.jss.2014.05.075>