Pseudonymization as a Service: Compartmentalizing & Controlling Data Processing in Evolving Systems with Micropseudonymization

Job Doesburg[1,*], Bernard van Gastel[1] and Erik Poll[1]

[1]NOLAI, Radboud University, Erasmusplein 1, Nijmegen, The Netherlands

Abstract

IT systems can be complex, processing data for different purposes in different subsystems, e.g. in microservice or service-oriented architectures. With such growing complexity, the chance of compromise of one (sub)system increases. This comes with privacy risks. In this article, we demonstrate the strength of blind, polymorphic, transitive and distributed pseudonymization to cryptographically compartmentalize and control data, limit the impact of data breaches and preserve privacy during system evolution. While pseudonymization is a well-established technique to hide identities, we propose its application at much finer granularity to control data linkage between or even within microservices, which we call *micropseudonymization*. This extends functional compartmentalization to data compartmentalization, strengthening privacy. Specifically, we propose *Pseudonymization as a Service (PaaS)*: an architecture where different microservices process data about data subjects using different pseudonyms and communicate through a central pseudonymization service that monitors and controls data exchange. This allows for new subsystems to be added without (accidentally) violating privacy constraints, enabling privacy-preserving system evolution and implementing *privacy by design* by applying *pseudonymization by default*. While we explicitly propose a solution for microservice architectures, we believe our insight can be applied generally, for any data processing system with a functionally compartmentalized architecture.

Keywords

Compartmentalization, privacy, pseudonymization, microservices, privacy by design, software evolution

1. Introduction

Modern data processing systems can be complex. System architectures such as microservice or service-oriented architectures (SOAs), have enabled rapid growth of system functionalities. The rise of artificial intelligence, for example, drives the processing of data for additional purposes such as training or fine-tuning of AI models, and with the growing popularity of Software as a Service, an increasing number of legal parties are involved in data processing. Systems thus process data in an increasing number of subsystems, for different purposes, potentially by different parties (i.e. in different contexts [1]).

This growing complexity introduces privacy risks: the more subsystems are involved and thus the more complex the whole system grows, the higher the chance that eventually, one will be compromised, either intentionally (i.e. actors deliberately violating policies) or unintentionally (e.g. data leakage as result of hacking or failing security measures [2].). We see the consequences in numerous large-scale data breaches. To improve privacy without changing *what* data is processed (i.e. without processing *less* data), compartmentalization can be applied. Microservice architectures and SOAs are a form of *functional* compartmentalization, limiting the functional impact (i.e. the impact on the system's functioning, e.g. system integrity or availability) of compromise. The *data* they process, however, is often not actually compartmentalized due to the use of shared identifiers. Data can therefore be linked with other public data or data in other microservices. This may result in cascading privacy problems: data may appear in a different context [3] than intended (context collapse), and through combination of data (aggregation), new information may be revealed from patterns, exceeding the sum of parts [4], possibly even leading to (re)identification through quasi-identifiers [5].

BENEVOL'25: Belgium-Netherlands Software Evolution Workshop, November 17–18, 2025, Enschede, NL

*Corresponding author.

job.doesburg@ru.nl (J. Doesburg); bernard.vangastel@ru.nl (B. van Gastel); erik.poll@ru.nl (E. Poll)

0009-0004-4120-6977 (J. Doesburg); 0000-0002-0974-4634 (B. van Gastel); 0000-0003-4635-187X (E. Poll)

In this article, we demonstrate the strength of central, blind, polymorphic, transitive and distributed pseudonymization to compartmentalize data (in addition to functional compartmentalization), minimizing the privacy impact of (sub)system compromise, and enable privacy-preserving system evolution. While pseudonymization is a well-established technique to prevent identifiability of data subjects, we propose its application at much finer granularity to limit and control data linkage between different data compartments, formed by microservices. We call this *micropseudonymization*.

As we gradually explain in this article, this idea comes down to the following: where pseudonyms are normally only used to *hide* direct identifiers, micropseudonymization goes beyond this purpose and additionally aims to *separate* data processing in small isolated compartments, *minimize* the data processed in each compartment and *control* data linkage between them to centrally and cryptographically *enforce* specific privacy policies.[1] Or simply put: when each microservice uses its 'own' pseudonyms, unauthorized linkage of data across microservices is prevented by default, even when multiple services are compromised. This achieves significant privacy features, while being generic and flexible, which makes micropseudonymization a practical technology to implement: it only modifies the identifiers being used in communication between microservices. This maximally limits data linkability and performs data minimization without changing the functioning of the system. We consider this technology especially useful in complex systems with complex trust boundaries, e.g. involving different legal parties or actors in different roles that each may only see specific subsets of data.

Specifically, we propose *Pseudonymization as a Service (PaaS[2])*: an architecture where different microservices in a system processes data about the same data subjects using different domain-specific pseudonyms. Using a secure (i.e. blind, polymorphic, transitive and distributed) and central pseudonymization service, data can be converted and exchanged between these pseudonymization domains, while applying central monitoring and control on these conversions. The central authorities administering the pseudonymization service can enforce privacy policies that specify what data may be processed by which service. This helps to keep grip on data processing in complex and evolving systems and implements *privacy by design* by applying *pseudonymization by default*. We evaluate our ideas through a case study of a research data platform designed by the authors, that implements PaaS.

While the idea of using different pseudonyms for data in different domains to improve privacy is not groundbreaking and proposed earlier, for example for distributed governmental databases [7], doing it so granularly at the system architecture level and as a method to enable privacy-preserving system evolution in complex systems, has not yet been proposed to our current knowledge.

In order for micropseudonymization to be effective, we assume the data in each service to be strictly pseudonymous, meaning that they do not contain unique attributes (or combinations of them, called quasi-identifiers) shared with other services, that would allow for data linkage bypassing the pseudonymization service. Data should only be linkable via the identifiers. Applying micropseudonymization highly granularly, however, can actually be used to eliminate such quasi-identifiers in large parts of data processing, as we explain in subsection 3.5. We also do not consider timing and traffic analysis attacks, where data can be linked based on timing of events or other metadata. This typically holds for batch data processing in situations where the data subjects are not necessarily the 'users' of a system.

Terminology Throughout this article, we use the terms *system*, *subsystem*, *service* and *microservice* somewhat interchangeably for (software) systems that process data. Strictly speaking, a large data processing *system* can be composed of multiple *subsystems*, that all may also be considered as independent systems themselves, depending on the scope. For example, a large data processing system (perhaps even 'landscape') of a typical corporate organization may consist of different (sub)systems for customer relationship management, enterprise resource planning and billing, that all exchange data among each other. Some of these may be delivered *as a service* by a service provider. Others may be self-hosted and may even consist of more subsystems, such as a front-end, back-end and database service.

Typically, the term microservice is used to highlight the smallest self-contained unit of a system,

[1] Notice that *hide*, *separate*, *minimize*, *control* and *enforce* are all common privacy-design strategies [6].
[2] Not to confuse with *Platform as a Service*.

responsible for carrying out a single, well-defined data processing function, such as performing a specific computation, data transformation or doing 'data storage'. Systems following a microservice architecture consist of only such independent microservices, while SOAs typically group related data processing functions together in somewhat broader services. In this article, however, we do not focus on these differences and conveniently call each functionally compartmentalized unit of the system in scope a microservice, regardless of the size or complexity of the function they perform. This, for example, also includes externally delivered SaaS systems. Typical examples are separation in different software processes, in different Docker containers, on different physical machines, hosted by different legal parties or any other form of trust boundaries. Human actors manually processing data in some way could also be considered. We thus consider a model where a data processing system consists of a network of independent microservices, all performing a well-defined data processing function and exchanging data among each other. While we discuss microservices, our insights transfer to general data processing systems with various types of trust boundaries.

2. Background: identifiers and pseudonyms

We first provide a background on the usage of identifiers in data processing and different forms of pseudonymization (specifically PEP [8]), which forms the basis of our PaaS architecture.

Data processing systems rely on identifiers to organize and reference entities (or data subjects) within their data. Fundamentally, these identifiers serve two distinct functional purposes:

1. **Internal reference:** To distinguish entities *within* a specific dataset or system and *internally* link different data points or attributes relating to the same entity.
2. **External reference:** To refer to entities *across* different datasets or systems and *externally* link between representations of the same entity in multiple systems.

For internal reference purposes, the specific value of an identifier can be arbitrary, requiring only local uniqueness within the boundaries of that dataset or system, which we call the *domain*. For external reference, a higher degree of uniqueness is required to provide linkability across multiple domains. Notice that the definition of what is internal and external, depends on the scope at which we consider a system (see also section 1). For clarity, in the remainder of this article, we refer to *data* as the combination of *identifiers* (or *pseudonyms*), which have internal or external reference as their core purpose, and *attributes*, which are any data related to an identifier.

Identifiability A special case of external reference is (legal) identifiability, or the ability to reference natural persons, the strongest form of external reference.[3] This can be illustrated with a microservice architecture example. Internal reference applies within the boundaries of a single microservice. External reference applies between different microservices of the same system. However, not all microservices need to interface with data subjects (and hence identify them) directly. Microservices that do not interface with data subjects directly, but do need to communicate with each other, could thus use specific identifiers for external reference that do not need to be identifiable, i.e. pseudonyms. The definition of internal and external reference thus depends on the scope of and abstraction level at which we consider a system.

2.1. Pseudonymization

With pseudonymization, highly linkable, or even identifiable *identifiers* that allow for external reference are replaced by domain-specific *pseudonyms* with only limited linkability. Considering this has been done properly, i.e. there are no further linkable attributes in the data and pseudonyms are truly unlinkable,

[3] We follow the legal definition of identifiability. From a technical viewpoint, sometimes, any form of external reference is called identifiability, which is why these values are called *identifiers* (or *nyms*, from which *pseudonyms* are derived). We only use identifiability for direct reference to natural persons.

this prevents data in one *pseudonymization domain* from being linked with other data in other domains. Pseudonyms can only be linked across domains with knowledge of some secret *additional information* [9], i.e. the (reverse) pseudonym mapping. Linkability is thus reduced to secrecy of this information.

Notation A common way to denote a pseudonym for (id)entity $X \in \mathcal{I}$ in domain $A \in \mathcal{D}$ is $X_{@A}$ and the associated pseudonymization function for domain A, $P_A : \mathcal{I} \to A$, can be used to convert X into $P_A(X) = X_{@A}$. To recover a pseudonym, we can write $P_A^{-1}(X_{@A}) = X$.[4] However, for pseudonymization between different domains A and B as we consider in this article, we more generally denote $P : A \times \mathcal{D}_A \times \mathcal{D}_B \to B$ so that $P(X_{@A}, A, B) = X_{@B}$. Notably, in these cases for any entity x we have $P^{-1}(x, A, B) = P(x, B, A)$. We thus do not consider a distinct recovery function, but consider each identifier to be a pseudonym in its own domain. The initial value from which pseudonyms in other domains are derived will be called the 'origin'.

To denote that pseudonyms are truly unlinkable, we write $X_{@A} \perp X_{@B}$. More precisely, one could write $\perp_c$ to indicate computational unlinkability, for example based on the (decisional) discrete log problem ($\perp_{\text{DLP}}$), and $\perp_\infty$ to indicate information-theoretic security. For simplicity, however, we will not discuss these details and default to just $\perp$ in this article.

2.2. Properties of pseudonymization methods

There are various ways of computing pseudonyms, including randomized or counter-based mapping tables, cryptographic hash functions or Message Authentication Codes (MACs) and symmetric or asymmetric encryption [10]. They can have a variety of different properties of which we highlight the most interesting ones for privacy and security below.[5]

Randomness (pseudonym unlinkability) Generated pseudonyms are completely random and do not encode any other information, hence they are truly unlinkable. Counter-based pseudonyms, for example, violate this as they contain the order in which they are generated, which may reveal a relation, hence $X_{@A} \not\perp X_{@B}$.

Statelessness A pseudonymization method is stateless if its outcome does not depend on previously performed operations. This is relevant because stateless functions are typically easy to implement securely and scale well. While completely randomized mapping-table based pseudonymization can provide information-theoretic unlinkable pseudonyms (i.e. $X_{@A} \perp_\infty X_{@B}$), their internal mapping tables either need to be pre-generated (infeasible for large domains), or generated at run-time, resulting in a stateful pseudonymization function that does not fit the desired type (i.e. $P : A \times \mathcal{D}_A \times \mathcal{D}_B \times S \to B \times S$ instead of $P : A \times \mathcal{D}_A \times \mathcal{D}_B \to B$). Therefore, efficient stateless pseudonymization methods only offer computationally unlinkable pseudonyms.

Reversibility After pseudonymizing from domain A to domain B, it is possible to pseudonymize back to domain A (i.e. $P(P(X, A, B), B, A) = X$). Notably, we only consider *efficiently reversible* pseudonymization methods that require an inverse operation with the same efficiency as the regular operation. We thus exclude brute force approaches like *rainbow tables*, as these are generic attacks that are not specific to the pseudonymization method itself.

Transitivity Pseudonymizing from domain A to domain B and then from domain B to domain C without first reversing to domain A, yields the same result as pseudonymizing from domain A to domain C directly (i.e. $P(P(X, A, B), B, C) = P(X, A, C)$). Transitivity implies efficient reversibility.

[4] P^{-1} is sometimes denoted as recovery function R.

[5] For simplicity, we assume all pseudonymization methods to be *deterministic* and *collision-free*, though some non-deterministic methods with collisions do exist.

Distribution Pseudonymization is distributed when multiple parties need to be involved in the computation of a pseudonym. The benefit of distribution is that no single party possesses the actual pseudonym mapping and all parties need to be involved to convert a pseudonym. Consequently, the pseudonym mapping is less likely to leak while there is replicated monitoring and control. In fact, each pseudonymization method can be distributed by applying it multiple times in sequence, such as multi-tier mapping-tables where $P = P_2 \circ P_1$ (for 2 tiers).

Commutativity When pseudonymization is distributed over multiple parties (i.e. $P = P_2 \circ P_1$), the order in which the parties perform their operations does not matter (i.e. $P_2 \circ P_1 = P_1 \circ P_2$). Notice that, though every form of pseudonymization be distributed, only specific forms achieve commutativity. Commutativity allows for changing the order of parties, potentially limiting the linkability of intermediate values by these parties.

Blindness (homomorphism) Pseudonymization takes place blindly, thanks to homomorphic properties, without observing the identifiers that are being pseudonymized, i.e. pseudonyms are encrypted before they are sent through a pseudonymization process so the party performing pseudonymization does not observe the pseudonyms. The pseudonymization function $P = \mathsf{Dec} \circ P' \circ \mathsf{Enc}$ thus consists of an encryption and decryption function, as well as a homomorphic function P' on the encrypted pseudonyms.

Polymorphism (encryption randomness) Encryption is *polymorphic* (or randomized or probabilistic) if there are multiple unlinkable ciphertext representations of the same plaintext. For blind pseudonymization, this not only disables the party performing the actual pseudonymization from observing the pseudonym itself, but also whether it is pseudonymizing an entity it has already seen previously or not. This is sometimes also called multi-show unlinkability.

2.3. Encryption as pseudonymization method

From a cryptographic perspective, reversible pseudonymization can be considered as a form of deterministic (i.e. non-randomized) encryption of identifiers. The pseudonym mapping essentially forms the key used for encryption, called the *pseudonymization secret*. Vice versa, encryption methods can be used for pseudonymization, where the encryption key forms the *additional information*.

In contrast to how encryption is normally used, for pseudonymization, the decryption key should typically *not* be shared with the recipient, as this would allow them to undo the pseudonymization. Also, normally, ciphertext linkability is considered as an inherent vulnerability of deterministic encryption that do not use random values, nonces, salts, seeds or initial values. For pseudonymization, however, we deliberately use deterministic ciphertexts to form pseudonyms that link but hide the identifiers. Using different keys for different domains allows for pseudonym unlinkability between the different domains.

Blind pseudonymization can be considered as two-layer encryption, where the inner layer of encryption converts the origin identifier into a pseudonym (without the pseudonymizing party sharing the 'decryption key'), and the outer layer is applied to hide the origin identifier from the pseudonymizing party itself and securely send the pseudonym to the recipient (who *does* receive the decryption key). The first layer of encryption should be performed homomorphically and deterministically, resulting in the same 'ciphertext' every time. The second layer of encryption should be polymorphic.

2.4. PEP: Polymorphic Encryption and Pseudonymization

The PEP (Polymorphic Encryption and Pseudonymization) framework [8] is an advanced form of pseudonymization [10], that has all the properties of pseudonymization mentioned in subsection 2.2.[6] It uses homomorphic operations on malleable ElGamal ciphertexts to blindly convert encrypted pseudonyms

[6] Actually, some small extensions of the framework are required to securely implement transitivity and distribution, which are not discussed in this article.

from one domain into another, while using polymorphic encryption to make communication unlinkable for intermediate parties. The PEP framework has implementations in the 'PEP Responsible Data Sharing Repository' for medical research data [11], as well as the Dutch national eID system *DigiD Hoog* [12, 13].

Pseudonymization using PEP enables secure pseudonymization between many different domains, without introducing a privacy hotspot and without single points of failure. Additionally, PEP can be used for end-to-end encrypted *asynchronous* communication. This means that pseudonymization can happen on encrypted data, without the sender being involved (long after initial encryption), enabling encryption at rest. We do not discuss this further in this article.

3. Data compartmentalization through pseudonymization

Microservice architectures are a form of functional compartmentalization: services run isolated from each other in their own containers and are protected from interfering with each other. Therefore, when one service is compromised, this has only limited impact on the functioning of other services in the system. Regarding data, however, compromise of one compartment (e.g. data leakage in one service) does *not* necessarily have a limited privacy impact on data in other compartments: the impact of a data breach can extend beyond the loss of secrecy of only that data, through unauthorized data linkage with other public data or data in other services. As a result, data may appear in a different context than it was originally disclosed in, but more importantly, through combination (aggregation) with other data, new information may be highlighted, possibly even re-identifying otherwise anonymous data. Typically, there are no measures in place to maintain boundaries between compartments and prevent data linkage across them.

In this section, we first generally describe how data compartments can be formed, through the use of pseudonyms in different pseudonymization domains. We then discuss the usage of a central pseudonymization service to convert data between those domains, followed by the properties of pseudonymization methods required to do this securely. Finally, we describe how to apply this concept in microservice architectures, followed by more advanced use cases.

3.1. Defining data compartments

In order to implement actual data compartmentalization, data processing must be organized in such a way that compromise of data in one compartment does not have impact on data in other compartments. Therefore, linkability of data between different data compartments must be prevented. Specifically, distinguishing identifiers and attributes, this requires data compartments to be subsets of data with:

1. **unlinkable identifiers**: any identifiers (or pseudonyms) used for processing must be specific to that compartment and unlinkable to identifiers in other compartments.
2. **unlinkable attributes**: any attributes *shared* between compartments must be anonymous or unlinkable. Attributes or combinations of attributes that *are* linkable, may only be processed in a single compartment.

The challenge thus lies in smartly defining which subsets of attributes are processed together in individual microservices (see also subsection 3.4). We can then use pseudonymization to create unlinkable identifiers for the data subject for each of these subsets, to form compartments. Whenever data is exchanged between compartments, the identifiers are converted from the pseudonymization domain associated with the one compartment, to the pseudonymization domain of the other compartment.

Having defined the subsets of data that form compartments, we can describe what systems have access to that data compartment: all systems that are involved in the processing of that data in that specific pseudonymization domain. If any of these is compromised, we may need to consider the data in this data compartment to be compromised as well. This concept does not need to be limited to software systems, but could also include hardware, manual processes or people, similar to what is commonly called a Trusted Computing Base (TCB). We discuss this further in subsection 3.5.

Notice that regular (not blind) pseudonymization methods would violate our rules for data compartmentalization: the data compartment performing pseudonymization, contains linked identifiers in both pseudonymization domains. For now, we will briefly consider this as an exception and consider the pseudonymization process to be trusted. In subsection 3.3, however, we present how blind and polymorphic pseudonymization methods mitigate this.

Horizontal and vertical compartmentalization Notice that we specifically consider vertical compartmentalization, where different compartments contain different attributes (columns) about the same data subjects (rows). Strictly speaking, however, horizontal compartmentalization is also a form of compartmentalization, where different compartments consist of different sets of data subjects, or entries for data subjects, (rows) but with the same attributes (columns). There are, however, some interesting use cases for applying pseudonymization for horizontal compartmentalization. An architecture where one system processes data for one half of the data subjects, and another system processes data for the other half, for example, would be a form of horizontal compartmentalization. There is typically no pseudonymization required to enforce compartmentalization here, as there are typically no relations between different data subjects. A more advanced use case could be to record transaction logs of the same data subject under different pseudonym domains to break linkability.

3.2. Central pseudonymization as a service

A data processing system can only meaningfully function if data can be exchanged between its subsystems. To exchange data between two data compartments, a pseudonymization service is required, converting pseudonyms between two pseudonymization domains. When there are more than two domains between which pseudonyms need to be converted, there could be two distinct pseudonymization processes, one pseudonymizing from domain A to B, and another from domain B to C. This is, however, not a scalable solution as for n domains, $n-1$ pseudonymization processes are required of all processes need to communicate with each other.

An alternative to many different pseudonymization processes would be to centralize pseudonymization in a single service that performs all pseudonym conversions between all domains. This has a number of key benefits:

1. **pseudonymization key secrecy**: it is easier to properly secure a pseudonymization secret that is stored at a central place, than multiple pseudonymization secrets (or even multiple copies of the same secret) in multiple places.
2. **monitoring**: central pseudonymization is easy to monitor. Specifically, for example, brute force attacks that enumerate all pseudonyms are easier to detect and prevent.
3. **access control**: it is easy to enforce access control rules in a central place. This provides flexibility to change policies in a single place.

There is, however, one important reason why a central pseudonymization service is undesirable using regular pseudonymization methods: central pseudonymization forms a great privacy hotspot. A regular central pseudonymization process could observe all data being exchanged between different data compartments and would be a single point of failure for the confidentiality and unlinkability of data. This would effectively undo any positive privacy effects of data compartmentalization. Considering this central pseudonymization service as a trusted, as stated before, would be an unreasonable assumption.

3.3. Blind, polymorphic, transitive and distributed pseudonymization

Regular pseudonymization methods are a single point of failure for the confidentiality and unlinkability of the pseudonyms they convert. There are, however, pseudonymization methods that do not suffer from these problems. Specifically, pseudonymization methods that are *blind*, *polymorphic*, *transitive* and *distributed* do not form a single point of failure or privacy hotspot.

1. **blind**: First, *blind* pseudonymization methods do not suffer from the privacy hotspot problem, because data is encrypted before it is sent through the pseudonymization service. The pseudonymization service itself does not have access to any data and is thus not part of any data compartment itself. There is thus no privacy hotspot.
2. **polymorphic**: In particular, *polymorphic* blind pseudonymization is required to prevent the pseudonymization service from linking the same pseudonym over different conversions.
3. **transitive**: Pseudonymization should be *transitive* when there are many compartments and many data flows between them. Without transitivity, pseudonymization from domain A to C (or C to A) must always go through domain B, causing linkability there.
4. **distributed**: Finally, it is desirable to perform pseudonymization in a *distributed* way, i.e. by multiple parties to eliminate single points of failure. Even if pseudonymization is performed blindly, it is still based on a pseudonymization secret which can leak. If in a n-tier distributed pseudonymization process one party leaks their pseudonymization secret, this does not directly break pseudonym unlinkability. Only if all n parties would be compromised, pseudonym unlinkability breaks. Additionally, *commutativity* may be a desirable property for practical implementations, but it does not prevent any particular privacy or security threats. In the remainder of this article, for simplicity, we will discuss a single pseudonymization service, which internally could consist of multiple distributed subsystems that together form a single pseudonymization service.

As mentioned in subsection 2.4, the PEP pseudonymization has all these properties.

3.4. Micropseudonymization

The idea of micropseudonymization is to organize data processing in the smallest possible data compartments. Ideally, each data compartment precisely belongs to a single microservice, performing a single well-defined data processing function, and consists of precisely the data required for executing that function. Since each microservice performs a single function, they often process different sets of attributes as input or compute different attributes as output.[7] Therefore, each microservice could be considered to process its own subset of data and can form its own data compartment.

If this is the case, each compartment has proper data minimalization and the boundaries of our data compartments are perfectly aligned with the trust boundaries of the (functionally compartmentalized) microservices. This maximally limits the impact of compromise of each microservice.

Whenever data needs to be exchanged between microservices, this must go through the pseudonymization service. Transitivity guarantees that data from different microservices can be properly combined, but only if the pseudonymization service allows it according to its configured policies. This is especially useful if microservices request data *just in time* (only at the time they need it), as this enables the pseudonymization service to log whenever data was used by a specific microservice.

Toy example We consider a (highly simplified) toy example of an e-commerce system. This system consists of microservices for *user authentication*, *order fulfillment*, *payments* and *shipping*. Each microservice forms their own data compartment:

- The *user authentication* data consists of usernames and passwords, stored under a user ID in the user-authentication-domain U $uid_{@U}$.
- The *order fulfillment* data consists of ordered products and parcel numbers, stored under a user ID in the order-domain O $uid_{@O}$.
- The *payment* data consists of bank transactions and payment confirmations, stored under a user ID in the payment-domain P $uid_{@P}$.
- The *shipping* data consists of payment confirmations and parcel numbers[8], stored under a user ID in the shipping-domain S $uid_{@S}$.

[7] An exception to this could be databases, which we discuss in subsection 3.5.

[8] As explained later, this may actually be a violation of data compartmentalization if the parcel number is unique.

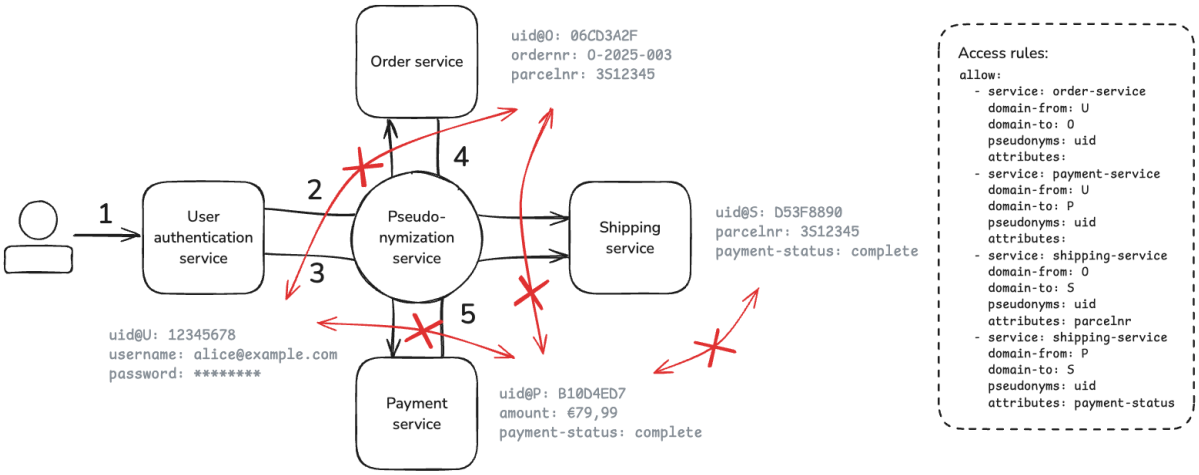

Figure 1: A toy example of an e-commerce system implementing PaaS, where data cannot be linked between the different microservices (in red) as $uid_{@U} \perp uid_{@O} \perp uid_{@P} \perp uid_{@S}$.

A typical flow in this system may look as follows (see also Figure 1).

1. To authenticate, user X provides their username and password to the user authentication service.
2. Upon placing an order, their associated user ID $X_{@U}$ is converted through the pseudonymization service and sent to the order service, who receives $X_{@O}$ to use in the fulfillment of the order.
3. Meanwhile, $X_{@U}$ is also sent to the payment service that uses $X_{@P}$ to process the user's payment.
4. After payment, the payment service sends a payment confirmation to the shipping service, converting $X_{@P}$ into $X_{@S}$.
5. The order service may send the parcel number to the shipping service, converting $X_{@O}$ into $X_{@S}$. The shipping service now has both the payment confirmation (from the payment service) and parcel number (from the order service) and knows it can ship the parcel.

Notice that except for the authorized data flows (see the access rules in Figure 1) that go through the pseudonymization service, data cannot be linked. The payment service does not know what was ordered, the order fulfillment service does not know anything about payments and no service except the user authentication service knows the user's identity. When the data in any of these microservices leaks, they remain unlinkable to other data in other microservices. An exception is the shipping service and the order service that share a parcel number and could combine their data, assuming that the shared parcel number is unique (which would indeed violate the unlinkable attributes assumption of section 3).

We could extend this scenario with a customer support service. This service could be allowed to request data from all different microservices, converting all those pseudonyms into a customer support pseudonym $uid_{@C}$. In order to allow this, however, the pseudonymization service could implement a policy where a customer support agent first needs to justify their request through a customer support ticket before pseudonymization is allowed to happen. This architecture can thus also be used to enforce more complex privacy policies, instead of just static ones.

3.5. Advanced use cases

So far, we considered data compartments to belong to a single microservice to minimize the amount of data processed in that compartment and limit the attack surface. The reverse, however, does not need to be true. There are actually good reasons for a single microservice to operate in multiple pseudonymization domains and multiple compartments (see also vertical compartmentalization in subsection 3.1).

For example, a survey system may process different surveys for the same set of respondents. When using a single pseudonymization domain, their responses to different surveys would be linkable within that survey system. By associating each survey with its own pseudonymization domain and applying

pseudonymization between them, linkability of surveys can be prevented. The system will thus process responses to survey 1 under $X_{@survey1}$ and to survey 2 under $X_{@survey2}$. For true unlinkability in case of compromise, it is important that the survey system itself is not allowed to convert pseudonyms between the different pseudonymization domains.

A similar construction could even be used in our e-commerce system to make different orders of the same customer unlinkable to each other within the order fulfillment service, with different pseudonymization domains per month, week or day (or even order number). By applying this thoroughly and at a fine-grained level, every individual attribute could be stored under a different pseudonym. Only when two attributes really need to be combined for a specific computation, their pseudonyms can be converted *just in time* by the pseudonymization service for the short duration of that computation. This way, even quasi-identifiers can be eliminated during significant parts of data processing.

Database example A concrete example of this concept could a stateful microservice that covers multiple trust and business logic boundaries, such as a central database service. Many microservice architectures depend on a single central database that stores the data that all other microservices process. When all this data is stored as a single data compartment, this would obviously be a great privacy hotspot. Applying micropseudonymization *within* this database service, however, could change this.

We could store the data for each microservice in different tables with different pseudonymization domains under different pseudonyms. The database itself does not need to be able to convert between the different pseudonymization domains, it just needs to store the data and perform simple queries. In this case, the database would have access to all data compartments and store their data, but is not able to link data between these compartments (only the services that depend on it). Notice that in this case, microservices are not able to exchange data with each other *via* this database or perform queries across compartments. A different middleware layer that integrates the pseudonymization service is required to facilitate this properly.

Micropseudonymization can thus be applied to separate data processing both *between (inter)* different microservices, and *within (intra)* a single microservice. By choosing pseudonymization domains smartly, micropseudonymization can be used to perform data minimalization, even within a single microservice.

4. Case study: research data platform

As presented in section 1, maintaining privacy in complex and evolving systems can be challenging. In this section, we present a specific use case where these challenges are especially relevant, and where micropseudonymization can be used.

Authors are involved in the development of a research data platform for the Dutch National Education Lab AI (NOLAI) at Radboud University that implements the PaaS architecture. Within NOLAI, many design-based research projects (10 new 3-year projects each year) are being conducted in cocreation between schools, researchers and businesses. Within these projects, prototype applications involving AI are being designed, developed, tested and scientifically validated within schools. These projects involve various forms of data processing in various different contexts.

For example, *operational* data for the primary functioning of the prototype is typically processed by businesses and schools. Meanwhile, parts of the data may be processed for *training* of AI models or general product development, by researchers and businesses. Finally, *scientific validation* data is collected to validate the educational effectiveness of the prototype. This may involve selected parts of the operational data, but may also involve additional data, such as survey data or background information about a student's learning deficiencies. Even more so, there may actually be several different research studies being conducted by different independent researchers, analyzing different subsets of the operational, training and scientific validation data.

In order to maintain privacy, it is crucial that these different forms of data are securely processed and properly governed. Survey data containing sensitive background information about a student's nationality, collected for scientific validation, for example, must be prevented from ending up in trained

AI models. Meanwhile, some data may *not* be shared with the researchers, while other data may *only* be shared with the researchers and not with the school. There is thus a complex system of authorized data flows between (sub)systems, hosted by different parties involved in these projects. Data is not processed in just a single research context, but in multiple different operational, product development and research contexts. Regular pseudonymization (replacing a direct identifier like a name for a single pseudonym that stays the same throughout all systems), as usual in scientific research, therefore does not suffice, as it does not prevent linkage of the different types of data between different systems.

System architecture The research data platform consists of several autonomous (sub)systems that can exchange data between each other. Examples include services for participant registration, surveys, file uploads, dashboards, long-term data storage, and various data analysis systems or integrations with prototype applications. They all have different business logic and trust boundaries between them, and are all involved in different research projects. With new projects starting every few months, the platform is always evolving and new systems are integrated frequently.

For each project, different pseudonymization domains are defined and associated with these 'users' (both functional systems, or individual researchers) that may get access to subsets of data from other 'users'. In principle, each service processes data with pseudonyms in its own domain. This does not need to effect the internal functioning of these services. For example, authors have integrated an existing LimeSurvey service as a data source via a API wrapper that only converts encrypted pseudonyms.

Only whenever data is exchanged between services, this goes through the central pseudonymization service (further just called PaaS). For example, when a research coordinator decides to start a survey, data from the participant registration service is sent to the survey system via PaaS and when a survey response is received for a specific participant, a notification is sent to the research progress dashboard, also via PaaS. And when a researcher then wants to analyze the survey responses together with log data uploaded to the file upload service, this also goes through PaaS. Each of these data flows must be explicitly authorized by PaaS, which only requires a simple configuration file to allow or disallow specific data flows. This centralizes data governance even though the systems themselves are decentralized.

Characteristics The PaaS architecture allows for rapid development, where different (micro)services can easily be developed and integrated while preserving privacy. We identify a few characteristics of this use case. We believe our insights hold generally for any systems where the architectural complexity and evolution of data processing may cause privacy challenges.

1. **High chance of compromise**: Because of the nature of cocreation, the variety and large number of projects, there is a wide variety of parties, systems and actors involved in the data processing. For each project, new data processing systems need to be set up (due to the nature of the projects) and the development speed of systems is high. Therefore, there is a high chance that at some point in time, one of these parties, systems or actors will be compromised (again, either intentionally or unintentionally) and their data is leaked. Since the *chance* is high, one can only maximally limit the *impact* of compromise to minimize privacy risk ($risk = chance \times impact$). Micropseudonymization does exactly this: it maximally limits the impact of data breaches without actually changing the attributes being processed in a specific microservice.
2. **Generic and practical solution**: Micropseudonymization is a generic privacy enhancing technology that can be implemented regardless of what form data processing actually takes within the individual microservices. It can be applied whenever a system has some form of functional compartmentalization and data exchange between compartments is properly implemented. This is important for this use case, because of the variety, scale and rapid development speed of systems. It is not feasible to spend significant time on implementing complex privacy-preserving computations that are specific to a single project. Micropseudonymization is a generic and practical measure, that is able to interface with both newly developed and existing systems.
3. **Centralized data governance**: Data governance suffers from a complex data processing architecture. When data is processed in numerous systems, with decentralized access control in each

of them, it becomes difficult to keep track of who may have had access to what data. Managing access rules and monitoring who accessed which data is challenging and error-prone when it is not centrally organized. By applying PaaS, access control on data linkability can be centralized despite the system architecture being highly decentralized.

A similar approach of pseudonymization using the PEP framework has been implemented in a long-term large-scale Parkinson's Disease Study [14], to disclose pseudonymous subsets of medical data to different research groups worldwide. Here, the system architecture itself is fixed and does not evolve, but the users and the datasets they request do evolve. In our research data platform, not only the users, but also the system architecture itself evolves. In both cases, central pseudonymization can be used to keep grip of data processing during evolution.

5. Future work

Data subject authentication So far, we discussed systems in which different microservices exchange data *about* data subjects with each other using pseudonyms, but we did not consider users communicating with microservice *themselves* (using pseudonyms). Whenever this needs to happen (e.g. our toy example in subsection 3.4), the user-interfacing service needs to authenticate the user. Extensions are required to enable the pseudonymization service to do this, similar to the goal of anonymous credential systems or typical pseudonym systems [15, 16, 17].

Pseudonymization service integrity While the PEP framework does realize privacy-friendly conversion of pseudonyms, it does not yet guarantee integrity or authenticity. Ongoing research by the authors, however, shows promising results using zero-knowledge proofs to enforce integrity throughout the pseudonymization process, where different distributed transcryptors can blindly verify each other that pseudonymization is performed correctly.

Distributed tracing Modern microservice architectures implement extensive monitoring tools to deal with the architectural complexity. An example is distributed tracing, where events or request are traced through different microservices, to see what goes wrong. These traces, though useful for debugging purposes, can be a big problem for privacy, as they enable data linkage and could even include user IDs directly. Pseudonymization of trace IDs, making them also domain-specific, may be an interesting approach for privacy-friendly distributed tracing, where trace IDs can only be linked through a pseudonymization service.

6. Conclusion

Privacy and security are cross-cutting concepts: they can not easily be added to a system but need to be implemented throughout the whole design. At its core, micropseudonymization solves the fundamental challenge that privacy becomes harder to maintain as system architectures become more complex. Traditional approaches to incorporate privacy may fail when system boundaries change. Micropseudonymization aligns privacy boundaries with security trust boundaries and business logic boundaries that microservice architectures already implement, implementing *privacy by design* by applying *pseudonymization by default*.

Within these compartments, micropseudonymization remains flexible and generic: the internals of a microservice are not adapted when micropseudonymization is implemented, except for some middleware to exchange data with other services. This makes micropseudonymization itself a generic and reusable component. It can be implemented in any well-designed data processing system that implements some form of functional compartmentalization, to limit the impact of data breaches. Moreover, by centralizing data governance, PaaS can be used to control and regulate data exchange to improve system monitoring, and to enforce important privacy features in complex and evolving data processing systems.

Acknowledgments

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

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] J. Doesburg, P. van Gastel, B. van Gastel, E. Poll, Data processing diagrams - A modeling technique for privacy in complex data processing systems, in: F. Bieker, S. D. Conca, J. M. del Álamo, Y. S. Martín (Eds.), Privacy and Identity Management. Generating Futures - 19th IFIP WG 9.6/11.7 and IFIP WG 11.6 International Summer School, Privacy and Identity 2024, Madrid, Spain, September 10-13, 2024, Revised Selected Papers, volume 705 of *IFIP Advances in Information and Communication Technology*, Springer, 2024, pp. 115–131. URL: https://doi.org/10.1007/978-3-031-91054-8_6. doi:10.1007/978-3-031-91054-8_6.

[2] C. Meijer, B. van Gastel, Self-encrypting deception: Weaknesses in the encryption of solid state drives, in: 2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019, IEEE, 2019, pp. 72–87. URL: https://doi.org/10.1109/SP.2019.00088. doi:10.1109/SP.2019.00088.

[3] H. Nissenbaum, Privacy in Context Technology, Policy, and the Integrity of Social Life, Stanford University Press, 2009.

[4] D. J. Solove, A taxonomy of privacy, University of Pennsylvania Law Review 154 (2006) 477–564. doi:10.2307/40041279.

[5] L. Sweeney, Simple demographics often identify people uniquely, Carnegie Mellon University, Data Privacy (2000). URL: https://dataprivacylab.org/projects/identifiability/paper1.pdf.

[6] J.-H. Hoepman, Privacy design strategies, in: ICT Systems Security and Privacy Protection, IFIP Advances in Information and Communication Technology, Springer, 2014, pp. 446–459. doi:10.1007/978-3-642-55415-5_38.

[7] J. Camenisch, A. Lehmann, (Un)linkable pseudonyms for governmental databases, in: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, ACM, 2015, pp. 1467–1479. doi:10.1145/2810103.2813658.

[8] E. Verheul, B. Jacobs, Polymorphic encryption and pseudonymisation in identity management and medical research, Nieuw Archief voor Wiskunde 18 (2017) 168–172.

[9] European Data Protection Board, Guidelines 01/2025 on Pseudonymisation, 2025. URL: https://www.edpb.europa.eu/our-work-tools/documents/public-consultations/2025/guidelines-012025-pseudonymisation_en.

[10] European Network and Information Security Agency, Pseudonymisation techniques and best practices: recommendations on shaping technology according to data protection and privacy provisions, 2019. URL: https://data.europa.eu/doi/10.2824/247711.

[11] B. E. Van Gastel, B. Jacobs, J. Popma, Data protection using polymorphic pseudonymisation in a large-scale parkinson's disease study, Journal of Parkinson's Disease 11 (2021) S19–S25. doi:10.3233/JPD-202431.

[12] Logius - Ministerie van Binnenlandse Zaken en Koninkrijksrelaties, Privacy Impact Assessment DigiD Hoog, 2017. URL: https://www.eerstekamer.nl/overig/20190129/privacy_impact_assessments_pia.

[13] Logius - Ministerie van Binnenlandse Zaken en Koninkrijksrelaties, BSNk PP technische specificaties v11.1, 2024. URL: https://gitlab.com/logius/bsnk/bsnk-techspecs/bsnk/-/raw/main/BPTSlive.pdf.

[14] B. E. van Gastel, B. Jacobs, J. Popma, Data protection using polymorphic pseudonymisation in a large-scale parkinson's disease study, Journal of Parkinson's Disease 11 (2021) S19–S25. doi:10.3233/JPD-202431.

[15] D. Chaum, Security without identification: Transaction systems to make big brother obsolete, Communications of the ACM 28 (1985) 1030–1044. doi:10.1145/4372.4373.

[16] A. Lysyanskaya, R. L. Rivest, A. Sahai, S. Wolf, Pseudonym systems, in: Selected Areas in Cryptography, volume 1758 of *Lecture Notes in Computer Science*, Springer, 2000, pp. 184–199. doi:10.1007/3-540-46513-8_14.

[17] J. Camenisch, A. Lysyanskaya, An efficient system for non-transferable anonymous credentials with optional anonymity revocation, in: Advances in Cryptology — EUROCRYPT 2001, volume 2045 of *Lecture Notes in Computer Science*, Springer, 2001, pp. 93–118. doi:10.1007/3-540-44987-6_7.