Pseudonymization as a Service: Privacy-Preserving System Evolution through 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 and transitive pseudonymization, applied through a central pseudonymization service, 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*: 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, implementing *privacy by design* by applying *pseudonymization by default* and enabling privacy-preserving system evolution. While we explicitly propose a solution for microservice architectures, we believe our insights can be applied generally, for any data processing system with a functionally compartmentalized architecture.

Keywords

data compartmentalization, privacy, pseudonymization, micropseudonymization, microservices, privacy by design, system 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 entities 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 a 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 e.g. by anonymization), 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) of compromise. The *data* they process, however, is often not compartmentalized, because 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 its parts [4], possibly even leading to (re)identification through quasi-identifiers [5].

In this article, we demonstrate the strength of blind, polymorphic and transitive pseudonymization, applied through a central pseudonymization service, to compartmentalize data (in addition to functional compartmentalization), minimize 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*.

The 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. The key observation underlying this approach is that most data processing within a system does not require external reference to data subjects, only internal reference within a service, which unlinkable, domain-specific pseudonyms provide.

This buys substantial privacy while keeping the architecture and interfaces generic and flexible, which makes micropseudonymization practical to implement: it modifies only the identifiers used in communication between services. This maximally limits data linkability and performs data minimization without changing the internal functioning of the system. We consider this technology especially useful for complex and evolving 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.

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)

© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

[1]Notice that *hide*, *separate*, *minimize*, *control* and *enforce* are all common privacy-design strategies [6].

Specifically, we propose *Pseudonymization as a Service*: an architecture where different services in a system process data about the same data subjects using different domain-specific pseudonyms. Using a secure (i.e. blind, polymorphic and transitive) and central pseudonymization service, data can be converted and exchanged between pseudonymization domains, while applying central monitoring and control on these conversions. To further eliminate single points of failure, this service can internally be distributed over multiple parties.[2] The central authorities administering the pseudonymization service can enforce policies that specify what data may be processed by which service. This helps to govern 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 this architecture.

The idea of using different pseudonyms for data in different domains to control linkability is well established and has been proposed before, for example for distributed governmental databases [7] and for pseudonymization of data lakes [8]. These works, however, focus on pseudonym conversion as a cryptographic primitive rather than as an architectural pattern. The same idea is also widely deployed in identity management, where pairwise or sector-specific pseudonyms give each relying party a different identifier for the same user, although there the separation is confined to the identity layer. Our contribution is to apply pseudonymization at the granularity of individual microservices, with a central service mediating and governing all data exchange, specifically as a method to enable privacy-preserving system evolution. We discuss this related work in detail in Section 5.

In order for micropseudonymization to be effective, we assume the data in each service to be strictly pseudonymous, meaning that it does not contain quasi-identifiers [5] shared with other services that would allow data linkage bypassing the pseudonymization service. We refer to this as the *unlinkable attributes* assumption, and make it precise in Section 3.1. Assessing whether a given data partitioning satisfies this assumption depends on the specific data and adversary model, which is beyond the scope of this article. We do note, however, that applying micropseudonymization sufficiently granularly can itself help eliminate such quasi-identifiers, as we discuss in Section 3.7. We also do not consider timing and traffic analysis attacks, where data can be linked based on timing of events or other metadata. Ignoring these attacks is reasonable for batch data processing, where 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 systems that process data, which are typically but not exclusively software-based. A large data processing *system* may consist of multiple *subsystems*, each of which can itself be considered an independent system depending on scope. We use the term *microservice* broadly: any functionally compartmentalized unit responsible for a well-defined data processing function, regardless of its actual size or complexity. This includes traditional microservices, broader SOA services, externally delivered SaaS systems, or even human actors performing manual data processing. Typical examples of trust boundaries between them are separation in different software processes, Docker containers, physical machines, or hosting by different legal parties. We consider a data processing system as a network of such microservices exchanging data among each other. While we use microservice terminology, our insights transfer to any system architecture with distinct trust boundaries.

[2] Distribution here refers to the internal implementation of the service, not to its architectural role: the service remains a single, central point of policy enforcement and monitoring, even when its computation is distributed over multiple parties.

2. Background: identifiers and pseudonyms

We first provide a background on the usage of identifiers in data processing and different forms of pseudonymization that form the basis of our 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. The definition of what is internal and external depends on the scope at which we consider a system. 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. 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.

2.1. Pseudonymization

With pseudonymization, highly linkable or even identifiable *identifiers*, or *nyms*, that allow for external reference are replaced by domain-specific *pseudonyms* with only limited

[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*. We only use identifiability for direct reference to natural persons.

linkability. Provided this is done properly, i.e. the data contains no linkable attributes and the pseudonyms are truly unlinkable, pseudonymization prevents data in one *pseudonymization domain* from being linked with 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 $P_A : \mathcal{I} \to A$ can be used to convert X into $P_A(X) = X_{@A}$. To recover the original identifier, 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 $P(X_{@A}, A, B) = X_{@B}$. In these cases, for any pseudonym x in domain B, 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 Diffie–Hellman assumption ($\perp_{\text{DDH}}$), and $\perp_\infty$ to indicate information-theoretic security. For simplicity, we default to just $\perp$ in this article, except where we state the assumption explicitly.

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]

Pseudonym unlinkability Generated pseudonyms do not encode any information beyond referencing the entity, making them unlinkable across domains. 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. Therefore, efficient stateless pseudonymization methods only offer computationally unlinkable pseudonyms.

[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.

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, together with the natural assumption that $P(X, A, A) = X$, implies efficient reversibility.

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. Every pseudonymization method can in principle be distributed by applying it multiple times in sequence, such as multi-tier mapping tables where $P = P_2 \circ P_1$ (for two 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 can 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 Pseudonymization takes place blindly, without the pseudonymizing party observing the identifiers being pseudonymized: pseudonyms are encrypted before they are sent through the pseudonymization process. This can be achieved through homomorphic properties of the encryption scheme: the pseudonymization function $P = \mathsf{Dec} \circ P' \circ \mathsf{Enc}$ consists of an encryption and decryption function, as well as a homomorphic function P' on the encrypted pseudonyms.

Polymorphism Encryption is *polymorphic* (or randomized or probabilistic) if there are multiple unlinkable ciphertext representations of the same plaintext. For blind pseudonymization, this prevents the party performing the pseudonymization from observing the pseudonym, and also from telling whether it has pseudonymized that entity before. This is sometimes also called multi-show unlinkability.

2.3. Pseudonymization by encryption

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. Normally, ciphertext linkability is considered an inherent vulnerability of deterministic encryption, which by definition uses no nonces, salts or initialization vectors. For pseudonymization, however, we deliberately use deterministic ciphertexts to form pseudonyms that hide the identifiers. Using different keys for different domains allows for pseudonym unlinkability between the different domains.

Blind pseudonymization can be understood as two layers of encryption. The outer layer is polymorphic and hides the origin identifier from the pseudonymizing party, who does not receive the decryption key for this layer. The pseudonymizing party instead applies the pseudonymization function homomorphically through this outer layer, converting the encrypted pseudonym from one domain to another without observing it. The recipient receives the decryption key for the outer layer and decrypts it to obtain the pseudonym, but does not receive the pseudonymization key, preventing them from undoing the pseudonymization. The pseudonymization itself is deterministic, resulting in the same pseudonym every time, while the outer encryption is polymorphic, ensuring unlinkability across conversions.

2.4. PEP: Polymorphic Encryption and Pseudonymization

The PEP (Polymorphic Encryption and Pseudonymization) framework [11] is a pseudonymization method based on malleable ElGamal encryption. It uses homomorphic operations on ElGamal ciphertexts to pseudonymize without observing the pseudonyms being converted (blindness), while rerandomization of ciphertexts prevents the converter from recognizing the same entity across different conversions (polymorphism). The PEP framework has been deployed in the PEP Responsible Data Sharing Repository for medical research data [12] and in the Dutch national eID system DigiD Hoog [13, 14]. The two paragraphs below add transitivity and distribution, giving the variant we use every property of Section 2.2 that our architecture requires.

Transitivity As originally defined [11], PEP only converts pseudonyms in a single direction: from a global *polymorphic pseudonym* (an encrypted origin identifier) to a domain-specific *local pseudonym*, using a single pseudonymization factor s for reshuffling. This operation is not (directly) reversible or transitive, as converting from domain A to B and then from B to C does not yield the same result as converting from A to C directly. Without transitivity, every conversion between two domains would need to pass through the origin domain, creating a linkability bottleneck that contradicts the compartmentalization goal. To achieve transitivity, the reshuffling factor can be replaced by $s = s_{\text{from}}^{-1} \cdot s_{\text{to}}$ [15], converting directly between local pseudonyms. No global pseudonym then exists anywhere in the system, and the pseudonymization service can convert directly between any two domains. This extension preserves the blindness and polymorphism properties of the original framework. It does, however, require identifiers to be mapped to group elements using a reversible encoding function such as Lizard [16], since the original PEP's use of a one-way hash-to-group suffices only when pseudonymization is unidirectional. We refer to this transitive variant throughout this article.

Distribution The PEP framework can be extended to distribute conversion over multiple transcryptors, each holding their own pseudonymization secret [15]. During conversion, each transcryptor applies its own reshuffling and rekeying operations in sequence, so that no single transcryptor possesses the complete pseudonymization mapping. Because these operations are scalar multiplications, they commute, so the order of transcryptors does not affect the result. Similarly, transcryptors may jointly provide users with their decryption key, preventing any single transcryptor from decrypting data on its own. This eliminates single points of failure for both pseudonym unlinkability and data confidentiality.

3. Data compartmentalization through pseudonymization

Microservice architectures are a form of functional compartmentalization: services run isolated from each other, for example 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: through unauthorized linkage with public data or with data in other services, the impact of a breach can extend beyond the loss of secrecy of the breached data itself.

As set out in Section 1, this can move data into a different context and, through aggregation, reveal new information or even re-identify 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 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, the properties of pseudonymization methods required to do this securely, and the access rules such a service enforces. Finally, we describe how to apply this concept in microservice architectures, including its implications for system evolution, state the resulting threat model, and give some 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 (i.e. nyms 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 unlinkable. Attributes

or combinations of attributes that *are* linkable (e.g. quasi-identifiers), may only be processed in a single compartment.

The challenge thus lies in carefully defining which subsets of attributes are processed together in individual microservices (see also Section 3.5). 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 or hardware (similar to what is commonly called a Trusted Computing Base), but could also include manual processes or people.

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 Section 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). 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. Typically no pseudonymization is required to enforce compartmentalization here, since there are usually no relations between different data subjects to hide. There are, however, some interesting advanced use cases for horizontal compartmentalization with pseudonymization. For example, different transaction logs or location history for a single data subject can be recorded in different pseudonym domains to limit 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, $\binom{n}{2} = \frac{n(n-1)}{2}$ pseudonymization processes are required if all domains 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 secure one pseudonymization secret stored in a central place than multiple secrets, or multiple copies of the same secret, spread across many places.
2. **Monitoring**: central pseudonymization is easy to monitor. Brute force attacks that enumerate all pseudonyms, for example, 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 when regular pseudonymization methods are used: it forms a 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.

3.3. Blind, polymorphic and transitive pseudonymization

The privacy hotspot problem identified above can be mitigated by requiring specific properties of the pseudonymization method. Specifically, pseudonymization methods that are *blind*, *polymorphic* and *transitive* do not form a privacy hotspot.

1. **Blindness**: 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.
2. **Polymorphism**: In particular, *polymorphic* blind pseudonymization is required to prevent the pseudonymization service from linking the same pseudonym over different conversions.
3. **Transitivity**: Pseudonymization should be *transitive* when there are many compartments and many data flows between them. Without transitivity, after pseudonymization from domain A to B, pseudonymization from domain B to C (or C to B) must always go through domain A, causing linkability there.

Additionally, it is desirable to perform pseudonymization in a *distributed* way, i.e. by multiple parties. Even if pseudonymization is performed blindly, it is still based on a pseudonymization secret which can leak. In an n-tier distributed pseudonymization process, pseudonym unlinkability only breaks if all n parties are compromised. Distribution is thus not required for correctness of the architecture, but strengthens it against key compromise. In the remainder of this article, for simplicity, we discuss a single pseudonymization service, which internally could consist of multiple distributed subsystems while retaining centralized policy enforcement and monitoring.

As mentioned in Section 2.4, PEP has all these properties.

3.4. Access control and information flow

A central pseudonymization service not only performs pseudonym conversion, but also enforces policies governing which conversions are permitted. These access rules can take various forms:

1. **Domain-based rules**: specifying which pseudonymization domains may be converted between, e.g. the payment service may receive data from the order service but not from the user authentication service.
2. **Role-based rules**: specifying which users or roles may request specific conversions, e.g. only a research coordinator may request data to be sent to a new analysis service.
3. **Conditional rules**: requiring additional justification or context before a conversion is authorized, e.g. a customer support agent must provide a support ticket number before accessing data across multiple services.
4. **Temporal rules**: limiting conversions to specific time windows, e.g. data may only be shared with an external auditor during the audit period.

These rules are configured centrally in the pseudonymization service, providing a single point of policy enforcement despite the decentralized system architecture.

Given a set of access rules, one can analyze the (*transitive closure* of) permitted data flows to derive system-level privacy properties. For example, one could verify that no sequence of authorized conversions allows a specific service to combine data from two compartments that should remain unlinkable, even considering compromise of specific parties. This enables reasoning about the privacy guarantees of a system configuration and is especially valuable when the system evolves: adding a new service or modifying access rules can be analyzed for its impact on existing privacy properties before deployment.

3.5. Micropseudonymization

The idea of micropseudonymization is to align data compartments with the trust boundaries of individual microservices. The feasibility of this approach rests on a key observation: in most microservice architectures, only a small number of services at the system boundary need to identify data subjects directly, e.g. for user-facing authentication or communication. The remaining services can process data using only internal references and never need to know *who* a data subject is. For these services, any unique identifier suffices; replacing shared identifiers with domain-specific, unlinkable pseudonyms therefore requires no change to their internal logic, only to the identifiers they exchange with other services.

In the simplest case, each data compartment 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, microservices often process different sets of attributes as input or compute different attributes as output. 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 minimization 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 when they need it), as it enables the pseudonymization service to log whenever data is 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 its own data compartment:

- The *user authentication* data consists of usernames and passwords, stored under a user ID $uid_{@U}$ in the user authentication domain U.
- The *order fulfillment* data consists of ordered products and parcel numbers, stored under a user ID $uid_{@O}$ and an order ID $oid_{@O}$ in the order domain O.
- The *payment* data consists of payment status for an order, stored under the order ID $oid_{@P}$ in the payment domain P.
- The *shipping* data consists of shipping status, stored under an order ID $oid_{@S}$ in the shipping domain S.

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

1. To authenticate, a user provides their username and password to the user authentication service.
2. Upon placing an order, their user ID $uid_{@U}$ is converted through the pseudonymization service and sent to the order service, which receives $uid_{@O}$ and generates an order ID $oid_{@O}$.
3. The order service sends $oid_{@O}$ to the payment service together with the order total; the payment service processes the payment and stores it under $oid_{@P}$.
4. After payment, the payment service sends a payment confirmation to the shipping service, converting $oid_{@P}$ into $oid_{@S}$.
5. The order service sends the order ID to the shipping service, converting $oid_{@O}$ into $oid_{@S}$. The shipping service now has both the payment confirmation (from the payment service) and the order reference (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, pseudonyms cannot be linked across microservices. The payment service does not know what was ordered or by whom; the order fulfillment service knows nothing 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.

Now suppose the order service also shares the parcel number with the shipping service, so that the shipping service

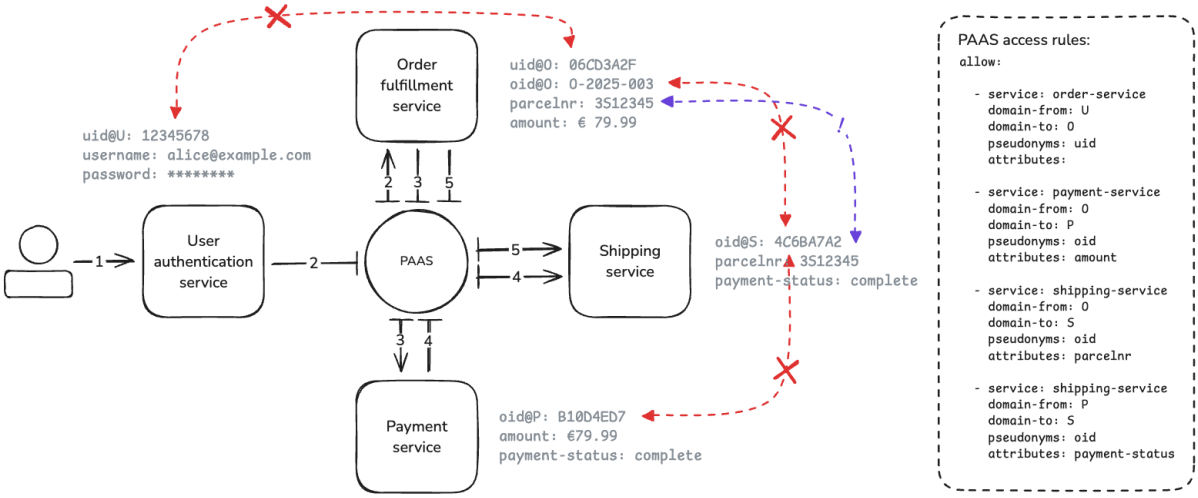

Figure 1: A toy example of an e-commerce system implementing micropseudonymization, where data cannot be linked between the different microservices as $uid_{@U} \perp uid_{@O}$ and $oid_{@O} \perp oid_{@P} \perp oid_{@S}$. The parcel number may act as a quasi-identifier.

can track the delivery. Since the parcel number is a unique attribute now present in both compartments, it acts as a quasi-identifier that enables data linkage between the order and shipping compartments, bypassing the pseudonymization service entirely. The unlinkable attributes assumption of Section 3.1 no longer holds for these two compartments. In practice, such shared attributes must either be eliminated, e.g. by pseudonymizing the parcel number as well, or accepted as an explicit and controlled relaxation of compartmentalization.

We could extend this scenario with a customer support service. This service could be allowed to request data from all different microservices, converting pseudonyms into equivalents $uid_{@C}$ and $oid_{@C}$ in the customer support domain C. Such access can be made conditional, as in the support-ticket rule of Section 3.4. This architecture can thus also be used to enforce more complex privacy policies, instead of just static ones.

System evolution This architecture is particularly well-suited for evolving systems. When a new microservice is added, it is assigned its own pseudonymization domain and the authorized data flows are configured in the pseudonymization service. Existing compartments and their privacy guarantees remain unaffected, as the new domain is unlinkable to all others by default. Privacy constraints can thus only be relaxed explicitly, by authorizing new conversions, never accidentally by adding new components. We demonstrate this in practice in Section 4.

3.6. Threat model

Having defined the architecture and its properties, we now make explicit what privacy guarantees micropseudonymization provides and under what assumptions.

We consider an adversary that compromises one or more microservices or the data they process. Compromise means the adversary obtains all data in the affected data compartment: both pseudonyms and attributes. We assume the pseudonymization service itself is not compromised, though it may be partially compromised in a distributed setting (i.e. fewer than all transcryptors are compromised). We do not consider timing or traffic analysis attacks (see Section 1).

Under this model, micropseudonymization provides the following guarantees:

1. **Compartment isolation**: compromise of compartment A reveals only the pseudonyms and attributes in A. Pseudonyms in A are computationally unlinkable to pseudonyms in any other compartment B under DDH in the group PEP instantiates (i.e. $X_{@A} \perp_{\text{DDH}} X_{@B}$), so the adversary cannot link records across compartments using pseudonyms alone.
2. **No cascading linkage**: even if the adversary compromises multiple compartments A and B, data in A and B remains unlinkable, provided the unlinkable attributes assumption holds (see Section 3.1).
3. **Policy-bounded combination**: the only way to link data across compartments is through the pseudonymization service, which enforces configured access rules (Section 3.4). An adversary who also obtains a compromised service's credentials can request the conversions that service is authorized for, but no others: the damage is bounded by the policy rather than by the compromise.

These guarantees degrade when the unlinkable attributes assumption is violated: if compartments A and B share a quasi-identifier, the adversary can link records regardless of pseudonym unlinkability. The transitive closure analysis described in Section 3.4 can be used to reason about which combinations of compromised compartments and authorized data flows could lead to unintended linkage. Formalizing this analysis is a direction for future work (Section 6).

3.7. Advanced use cases

So far, we aligned each data compartment with a single microservice. Micropseudonymization can, however, be applied at finer granularity: different subsets of data *within* a single microservice can be associated with different pseudonymization domains. This extends micropseudonymization from a technique for *inter*-service compartmentalization to one for *intra*-service data minimization. Attributes that do not need to be processed together do not need to be processed under the same pseudonym. By processing them

under different pseudonyms in different domains, they become unlinkable within that service, and can only be combined when explicitly authorized by the pseudonymization service. This directly addresses the unlinkable attributes assumption of Section 3.1: fine-grained pseudonymization can progressively eliminate quasi-identifiers by preventing unnecessary linkage of attributes.

The granularity at which this is applied can vary. At the coarsest level, different datasets within a service can be assigned different pseudonymization domains. For example, a survey system processing multiple surveys for the same respondents could associate each survey with its own domain, so that responses to survey 1 are processed under $X_{@\text{survey1}}$ and responses to survey 2 under $X_{@\text{survey2}}$, preventing linkability of responses across surveys. For true unlinkability in case of compromise, it is important that the pseudonymization service does not authorize the survey system to convert pseudonyms between these domains. At a finer level, temporal partitioning can prevent longitudinal linkability: in our e-commerce example, different orders of the same customer could use different pseudonymization domains per month, week or day, making order history unlinkable within the order fulfillment service. In the extreme case, every individual attribute could be processed under a different pseudonym, eliminating such quasi-identifiers entirely during storage. Attributes are then only combined under a shared pseudonym *just in time* by the pseudonymization service for the duration of a specific computation. This comes at the cost of increased pseudonymization service interactions and policy complexity, making it a trade-off between privacy granularity and operational overhead that must be tuned per deployment.

Database example A concrete application of intra-service micropseudonymization at the coarsest level is a central database shared by multiple microservices. When all data is stored in a single pseudonymization domain, the database forms a privacy hotspot. By storing the data for each microservice in separate tables under different pseudonymization domains, the database has access to all data compartments but cannot link data between them. Microservices cannot exchange data via the database or perform cross-compartment queries; a middleware layer integrating the pseudonymization service is required for this.

Micropseudonymization can thus be applied both *between* different microservices and *within* a single microservice. By choosing pseudonymization domains at the appropriate granularity, it serves as a mechanism for data minimization that goes beyond what functional compartmentalization alone can achieve.

4. Case study: research data platform

The authors are involved in the development of a research data platform that implements micropseudonymization, built for the Dutch National Education Lab AI (NOLAI) at Radboud University. Within NOLAI, many research projects are conducted in collaboration with schools, researchers and businesses. Within these projects, prototype applications involving AI are designed, developed, tested and scientifically validated. These projects involve many forms of data processing, in many 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 parts of the operational data, but may also involve additional data, such as survey data or background information about a student's learning deficiencies. Moreover, there may be several 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 a single research context, but in operational, product development and research contexts at once. Regular pseudonymization, as usual in scientific research, replaces a direct identifier such as a name with a single pseudonym that stays the same throughout all systems. This does not suffice: it prevents neither data linkage between contexts nor the emergence of quasi-identifiers.

System architecture The research data platform consists of several independent (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 may be integrated over time.

For each project, different pseudonymization domains are defined and associated with *users*, which may be either functional systems or natural persons such as individual researchers, and which may be granted access to subsets of data belonging to other users. In principle, each service processes data with pseudonyms in its own domain. This does not affect the internal functioning of these services.

Data goes through the central pseudonymization service only when it is exchanged between services. For example, when a research coordinator decides to start a survey, data from the participant registration service is sent to the survey system via the pseudonymization service and when a survey response is received for a specific participant, a notification is sent to the research progress dashboard, also via the pseudonymization service. When a researcher then wants to analyze the survey responses together with log data uploaded to the file upload service, this too goes through the pseudonymization service. Each of these data flows must be explicitly authorized by the pseudonymization service, 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.

Pseudonymization in scientific research This architecture is particularly well-suited for scientific research, where the distinction between identifiable and non-identifiable data processing is especially clear. Identifiability of data subjects is typically only required during a limited phase of research: for participant registration, informed consent, and initial data collection. Once data has been collected, the analysis phase operates entirely on pseudonymous data: researchers need to link records within their analysis context, but never need to know *who* a data subject is. Pseudonyms remain necessary during analysis, however, to support consent revocation, data removal requests, or follow-up in case of incidental findings, but these operations are mediated through the pseudonymization service without exposing identities to the researcher.

In some settings, data collection may gather a broader set of attributes than any individual analysis requires, with different researchers analyzing only different subsets of the collected data for different research studies. If only a single shared pseudonym is used throughout both phases and across all researchers, as is conventional in research data management, every researcher can link records across all collected attributes and across other researchers' datasets, even though they only need to link records within their own analysis context. Micropseudonymization avoids this: each researcher or analysis service receives only the attributes relevant to their study, under domain-specific pseudonyms unlinkable to those used during collection or by other researchers. Compromise of one researcher's dataset therefore does not affect the privacy of data held by others, even when derived from the same participants.

While these observations are specific to scientific research, the general principle applies broadly: data processing typically consists of different steps, phases, or actors at which data does not need to be linkable by default. Microservice architectures are a natural setting for micropseudonymization because they already organize these steps into functionally distinct services with well-defined boundaries.

Comparison with existing deployments A similar approach using the PEP framework has been implemented in a long-term large-scale Parkinson's Disease Study [12], disclosing pseudonymous subsets of medical data to different research groups worldwide. In that deployment, the system architecture is fixed and does not evolve, though the users and datasets they request do. In our research data platform, not only the users and datasets, but also the system architecture itself evolves: new services are integrated and new pseudonymization domains are defined on a regular basis. This represents a more challenging setting, as privacy guarantees must be maintained under architectural evolution, and our experience demonstrates that this is feasible in practice.

Characteristics The micropseudonymization 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 case study. We believe our insights hold generally for any system where the architectural complexity and evolution of data processing may cause privacy challenges.

1. **High chance of compromise**: The data processing involves a wide variety of parties, systems and actors in data processing, with new projects and systems being integrated on a regular basis. Because of the variety of involved parties and the high development speed, there is a high chance that at some point, one party will be compromised, either intentionally (deliberately using data for an unauthorized purpose) or unintentionally (e.g. as a result of hacking). Privacy risk is the product of *chance* and *impact*, so where the chance of compromise is high, the impact is the only remaining lever. Micropseudonymization pulls exactly that lever: it maximally limits the impact of data breaches without changing the attributes being processed in a specific microservice.
2. **Generic and practical solution**: The variety and rapid evolution of projects makes it infeasible to implement complex privacy-preserving computations tailored to each individual system. Micropseudonymization provides a generic measure that can interface with both newly developed and existing systems without modifying their internals.
3. **Centralized data governance**: When data is processed across numerous systems with decentralized access control, it becomes difficult to track who has access to what data. By applying pseudonymization as a service, access control on data linkability is centralized despite the system architecture being highly decentralized.

Observations The research data platform currently operates with various pseudonymization domains, covering different data points and attributes across multiple research projects. In practice, onboarding a new service requires a policy file edit to configure the authorized data flows and an API wrapper using middleware for pseudonym conversion. For example, integrating an existing LimeSurvey instance required only such a wrapper around its API, without any modifications to LimeSurvey itself.

We also encountered practical challenges. Configuring access rules requires careful attention: incorrectly configured policies can inadvertently block data flows, but this is precisely the kind of deliberate consideration that micropseudonymization is designed to enforce. Early implementations suffered from race conditions and deadlocks when many pseudonym conversions were performed concurrently, which were resolved through batch operations and careful concurrency management. Routing all data exchange through the pseudonymization service does impose architectural overhead that requires upfront consideration, but has proven feasible in practice.

5. Related work

The idea of using domain-specific pseudonyms to prevent cross-domain linkability has been explored in several contexts. We distinguish these along two axes: whether pseudonymization is performed blindly or in the clear, and whether it is performed by a service mediating data exchange or by an identity provider during the data subject's own authentication. Beyond pseudonymization itself, we also relate micropseudonymization to broader work on privacy engineering for software architectures.

Blind approaches Camenisch and Lehmann [7] proposed a protocol for unlinkable pseudonyms in governmen-

tal databases, where a central converter blindly converts pseudonyms between domains, with UC-security proofs. Lehmann extended this to ScrambleDB [8], which also uses the term pseudonymization-as-a-service but in a different architectural setting. ScrambleDB assumes a centralized data lake where all data is stored under per-attribute pseudonyms, fully unlinkable at rest, and selectively joined on demand for specific queries. Our architecture is more general in scope: rather than assuming a centralized data lake, it targets decentralized microservice architectures where autonomous services each store and process their own data and exchange it through the pseudonymization service. Both ScrambleDB and our architecture prevent unauthorized combination of data across domains through unlinkable pseudonyms: data is unlinkable by default and can only be combined through the central converter, though ScrambleDB produces ephemeral join pseudonyms per query while our architecture governs conversions through configurable access policies. Moreover, ScrambleDB assumes a fixed set of data sources and processors, whereas our architecture is designed for settings where new services are integrated on a regular basis.

Non-blind approaches Mainzelliste [17] is a widely deployed pseudonymization service that mediates data exchange for medical research, maintaining a central mapping table between identifiers and pseudonyms. It is, however, not blind: the service observes the complete identity-to-pseudonym mapping, forming a privacy hotspot. Micropseudonymization with blind pseudonymization eliminates this hotspot, as discussed in Section 3.3.

Domain-specific pseudonyms in identity management Using a different pseudonym per domain is an established and widely deployed idea in identity management. The OpenID Connect standard defines pairwise pseudonymous identifiers, where the identity provider issues a different subject identifier to each relying party or sector, so that relying parties cannot correlate users without cooperation [18]. This practice is recommended for federated identity by NIST [19], and some national eID schemes apply the same principle, such as the sector-specific personal identifier of the Austrian eID [20] and the Restricted Identification of the German identity card [21]. In most of these schemes the pseudonyms are derived non-blindly, by an identity provider during the data subject's authentication rather than by a service mediating data exchange, so the identity provider observes the resulting pseudonyms. Restricted Identification differs in that the sector-specific pseudonym is computed on the identity card itself, so no central party observes the mapping at all.

The Dutch BSNk infrastructure, used for the DigiD Hoog eID authentication service, is a notable exception: pseudonym conversion is performed by a central converter blindly using the PEP framework (see Section 2.4), so that the converter does not observe pseudonyms [13, 14]. OPPID [22] takes a similar blind approach for federated single sign-on: the identity provider derives the pairwise pseudonym blindly, learning neither the relying party it is derived for nor the resulting pseudonym.

In all of these, however, domain separation is limited to the identity layer: it only separates relying parties from one another. Each relying party still processes its data under a single pseudonym, with no further separation within the relying party itself and no true data compartmentalization. Micropseudonymization instead applies domain-specific pseudonyms both among and within services that do not need to identify data subjects at all.

Privacy engineering and architecture As a system architecture, micropseudonymization relates to a broader body of work on privacy by design and privacy engineering. Compartmentalization and least privilege are long-standing principles of security architecture [23]; micropseudonymization extends them from limiting the functional impact of compromise to limiting data linkability. Regarding privacy, our work can be understood as an architectural realization of the hide, separate, minimize, control and enforce strategies of Hoepman [6]. Spiekermann and Cranor distinguish privacy by policy from privacy by architecture [24]. Micropseudonymization combines both: data is unlinkable by architecture and minimized by default, while the central pseudonymization service architecturally enforces configurable access policy governing data linkage. Privacy threat modeling frameworks such as LINDDUN [25] elicit linkability and identifiability threats from the data flows of a system. Data processing diagrams [1] provide a related modeling technique for complex data processing systems with a focus on privacy. Where these approaches make linkability threats visible, micropseudonymization is a mechanism that mitigates them.

Privacy controls in microservice architectures Closer to our setting, recent work applies privacy controls within microservice architectures. Loechel et al. [26] integrate data minimization and purpose limitation directly into gRPC through interceptors that minimize data in response messages according to the access purpose, without modifying the service's internals. This is complementary to our approach: it minimizes the attribute values exchanged between services, where micropseudonymization makes the identifiers unlinkable. Its enforcement is also distributed, with one interceptor per service, whereas ours is concentrated at a single conversion point. In contrast to such per-service minimization techniques, micropseudonymization contributes a generic architectural mechanism that specifically controls data linkability, by construction, and preserves this guarantee as the system evolves.

6. Future work

The architecture as presented in this article leaves several directions for further research, both in strengthening the pseudonymization process itself and in extending the scope of micropseudonymization.

Quasi-identifier analysis As noted in Section 1, the effectiveness of micropseudonymization depends on the absence of linkable attributes across compartments. Developing systematic methods to analyze whether a given data partitioning contains quasi-identifiers, specifically in the presence of pseudonymized identifiers, is an important open problem. Such methods would be especially valuable for evolving systems, where each new service or data flow must be verified against existing compartmentalization guarantees.

Pseudonymization service integrity While the PEP framework realizes privacy-friendly conversion of pseudonyms, it does not yet guarantee integrity or authenticity of the conversion process. Ensuring that distributed pseudonymization services correctly perform their operations, without being able to observe the pseudonyms themselves, is a non-trivial problem that we are actively investigating using zero-knowledge proofs.

Data subject authentication We discussed systems in which different microservices exchange data *about* data subjects, but did not consider data subjects communicating with microservices *themselves*. Whenever this is required, the user-interfacing service needs to authenticate the data subject without revealing their identity to other services. Extensions are required to enable the pseudonymization service to facilitate such authentication, similar to the goals of anonymous credential systems [27, 28, 29].

Post-quantum pseudonymization The PEP framework relies on the hardness of the discrete logarithm problem on elliptic curves, which is vulnerable to quantum computing attacks. Investigating post-quantum alternatives, for example based on lattice assumptions such as Ring Learning with Errors (RLWE), that preserve the required properties (blindness, polymorphism and transitivity) is an important direction for the long-term viability of our architecture.

Privacy-preserving distributed tracing Modern microservice architectures implement distributed tracing to deal with architectural complexity, where events are traced through different microservices. These traces can be a problem for privacy, as they enable data linkage across services. Pseudonymization of trace IDs, making them domain-specific, may be an interesting approach for privacy-friendly distributed tracing.

7. Conclusion

Privacy and security are cross-cutting concerns: they cannot easily be added to a system but need to be implemented throughout the whole design. As system architectures grow in complexity and evolve over time, maintaining privacy guarantees becomes increasingly challenging. Traditional approaches to incorporate privacy are often tightly coupled to a specific architecture and may fail when system boundaries change, making privacy a concern that must be revisited with every architectural evolution. Micropseudonymization addresses this by aligning privacy boundaries with the trust boundaries and business logic boundaries that microservice architectures already implement, achieving *privacy by design* through *pseudonymization by default*.

This is made possible by the observation that most data processing within a system does not require identifiability of data subjects: only the services at the system boundary that directly interface with users need to identify them. All other services can operate on domain-specific pseudonyms without any loss of functionality. The conventional practice of using a single shared pseudonym throughout an entire system therefore exposes far more linkability than is functionally necessary. Micropseudonymization remains flexible and generic in addressing this: the internals of a microservice are not adapted, only the identifiers exchanged between services through middleware.

Crucially, this alignment between privacy boundaries and trust boundaries can be preserved as the system evolves. When new services are added to a system, they are assigned their own pseudonymization domain and data flows are configured through the central pseudonymization service. Privacy guarantees for existing services are not affected, as their compartmentalization remains intact. This makes micropseudonymization a practical enabler of privacy-preserving system evolution: systems can grow and change without inadvertently violating privacy constraints.

Our experience with the NOLAI research data platform demonstrates that this approach is practically feasible: new services can be integrated with minimal effort, and privacy guarantees are maintained as the architecture evolves.

The effectiveness of micropseudonymization does depend on proper partitioning of attributes across compartments to prevent quasi-identifiers from enabling unauthorized data linkage (see Section 3.1), though fine-grained pseudonymization can itself help eliminate such quasi-identifiers (see Section 3.7). Developing systematic methods for this analysis remains an important direction for future work. Nevertheless, by centralizing data governance through pseudonymization as a service, micropseudonymization provides a practical and enforceable mechanism to control data exchange and maintain privacy in complex and evolving data processing systems.

Acknowledgments

The research data platform described in Section 4 was developed by the first author together with Julian van der Horst and Harm van Stekelenburg. 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

During the preparation of this work, the authors used Claude Sonnet 4.5 and 4.6 (Anthropic) for paraphrasing and rewording, improving writing style, and grammar and spelling checks. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of this publication.

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: Privacy and Identity Management. Generating Futures – Privacy and Identity 2024, Springer, 2025, pp. 115–131. 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: IEEE Symposium on Security and Privacy – SP 2019, IEEE, 2019, pp. 72–87. 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, Data Privacy Working Paper 3, Carnegie Mellon University, 2000. URL: https://dataprivacylab.org/projects/identifiability/paper1.pdf.

[6] J.-H. Hoepman, Privacy design strategies, in: ICT Systems Security and Privacy Protection – SEC 2014, 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: ACM Conference on Computer and Communications Security – CCS 2015, ACM, 2015, pp. 1467–1479. doi:10.1145/2810103.2813658.

[8] A. Lehmann, ScrambleDB: Oblivious (Chameleon) pseudonymization-as-a-service, Proceedings on Privacy Enhancing Technologies 2019 (2019) 289–309. doi:10.2478/popets-2019-0048.

[9] EDPB, Guidelines 01/2025 on pseudonymisation, 2025. URL: https://www.edpb.europa.eu/our-work-tools/documents/public-consultations/2025/guidelines-012025-pseudonymisation_en.

[10] ENISA, Pseudonymisation techniques and best practices: Recommendations on shaping technology according to data protection and privacy provisions, 2019. doi:10.2824/247711.

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

[12] 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.

[13] Logius, Privacy impact assessment DigiD Hoog, 2017. URL: https://www.eerstekamer.nl/overig/20190129/privacy_impact_assessments_pia.

[14] Logius, BSNk PP technische specificaties v11.1, 2024. URL: https://gitlab.com/logius/bsnk/bsnk-techspecs/bsnk/-/raw/main/BPTSlive.pdf.

[15] J. Doesburg, B. van Gastel, E. Poll, n-PEP: Secure data sharing with transitive and distributed blind pseudonymization, in: Security and Trust Management – STM 2026, Springer, 2026. To appear.

[16] A. Westerbaan, L. Hendriks, Polymorphic encryption and pseudonymisation of IP network flows, in: IFIP Networking Conference – Networking 2020, IEEE, 2020, pp. 494–498. URL: https://ieeexplore.ieee.org/document/9142777.

[17] G. Tremper, T. Brenner, M. Ben Amor, T. Kussel, M. Lablans, Mainzelliste: Ten years of pseudonymization, record linkage, and informed consent management, Patterns 7 (2026) 101432. doi:10.1016/j.patter.2025.101432.

[18] N. Sakimura, J. Bradley, M. B. Jones, B. de Medeiros, C. Mortimore, OpenID Connect Core 1.0 incorporating errata set 2, OpenID Foundation specification, 2023. URL: https://openid.net/specs/openid-connect-core-1_0.html.

[19] D. Temoshok, J. P. Richer, Y.-Y. Choong, J. L. Fenton, N. Lefkovitz, A. Regenscheid, R. Galluzzo, Digital identity guidelines: federation and assertions, NIST Special Publication 800-63C-4, National Institute of Standards and Technology, 2025. doi:10.6028/NIST.SP.800-63C-4.

[20] H. Leitold, A. Hollosi, R. Posch, Security architecture of the Austrian citizen card concept, in: Annual Computer Security Applications Conference – ACSAC 2002, IEEE, 2002, pp. 391–400. doi:10.1109/CSAC.2002.1176311.

[21] J. Bender, Ö. Dagdelen, M. Fischlin, D. Kügler, Domain-specific pseudonymous signatures for the German identity card, in: Information Security – ISC 2012, Springer, 2012, pp. 104–119. doi:10.1007/978-3-642-33383-5_7.

[22] M. Kroschewski, A. Lehmann, C. Özbay, OPPID: Single sign-on with oblivious pairwise pseudonyms, Proceedings on Privacy Enhancing Technologies 2025 (2025) 629–649. doi:10.56553/popets-2025-0080.

[23] J. H. Saltzer, M. D. Schroeder, The protection of information in computer systems, Proceedings of the IEEE 63 (1975) 1278–1308. doi:10.1109/PROC.1975.9939.

[24] S. Spiekermann, L. F. Cranor, Engineering privacy, IEEE Transactions on Software Engineering 35 (2009) 67–82. doi:10.1109/TSE.2008.88.

[25] M. Deng, K. Wuyts, R. Scandariato, B. Preneel, W. Joosen, A privacy threat analysis framework: Supporting the elicitation and fulfillment of privacy requirements, Requirements Engineering 16 (2011) 3–32. doi:10.1007/s00766-010-0115-7.

[26] L. Loechel, S.-R. Akbayin, E. Grünewald, J. Kiesel, I. Strelnikova, T. Janke, F. Pallas, Hook-in privacy techniques for gRPC-based microservice communication, in: Web Engineering – ICWE 2024, Springer, 2024, pp. 215–229. doi:10.1007/978-3-031-62362-2_15.

[27] 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.

[28] A. Lysyanskaya, R. L. Rivest, A. Sahai, S. Wolf, Pseudonym systems, in: Selected Areas in Cryptography – SAC 1999, Springer, 2000, pp. 184–199. doi:10.1007/3-540-46513-8_14.

[29] J. Camenisch, A. Lysyanskaya, An efficient system for non-transferable anonymous credentials with optional anonymity revocation, in: Advances in Cryptology – EUROCRYPT 2001, Springer, 2001, pp. 93–118. doi:10.1007/3-540-44987-6_7.