n-PEP: Secure Data Sharing with Transitive and Distributed Blind Pseudonymization

Job Doesburg, Bernard van Gastel, and Erik Poll

NOLAI, Radboud University, Nijmegen, The Netherlands
{job.doesburg,bernard.vangastel,erik.poll}@ru.nl

Abstract. Organizations that share data about common data subjects typically rely on shared identifiers or a single trusted third party to convert between domain-specific pseudonyms. Such a party is a single point of failure: it must be fully trusted for data confidentiality and pseudonym unlinkability. We introduce n-PEP, an architecture for end-to-end encrypted, asynchronous, pseudonymized data sharing that follows the principle of *distributed trust* by spreading pseudonymization over n semi-trusted parties. Each party blindly converts pseudonyms and re-encrypts data, and can independently monitor and enforce data sharing policies, yet no party alone can read data or link pseudonyms. Confidentiality and unlinkability hold as long as at least one party remains uncompromised. n-PEP extends the PEP (Polymorphic Encryption and Pseudonymization) framework, used in the Dutch national eID infrastructure and in a (medical) research data repository, with reversible and transitive pseudonymization, enabling multidirectional data sharing. We identify a *linking-oracle* vulnerability in this repository, in which so-called polymorphic pseudonyms allow colluding users to link their pseudonyms, which n-PEP resolves by construction. We also present an open-source software library implementing n-PEP.

Keywords: Pseudonymization · Unlinkability · Data sharing · Proxy re-encryption · Distributed trust

1 Introduction

When organizations, such as government departments or hospitals and medical centers, want to exchange data about people, the basic approach is to use a shared unique identifier for each data subject, such as a social security number. From a privacy perspective this is not ideal: a global identifier makes all of a person's data trivially linkable, leaving data sharing effectively uncontrolled. Any party holding the identifier can link and combine records about a data subject, with no one able to oversee or restrict what is exchanged.

A privacy-friendly alternative is for different organizations (or groups of organizations) to use their own identifiers, which we call *pseudonyms*, such as different customer IDs at different companies. This compartmentalizes data [13] and pre-

vents uncontrolled data linkage across domains,[1] without irreversibly limiting data utility. Sharing data between domains then requires converting pseudonyms. Because every exchange must pass through this conversion, it forms a natural control point at which data sharing can be monitored and restricted. Conventionally, this relies on a single trusted converter. Even when pseudonyms are converted blindly, i.e. using encrypted values, this converter remains a single point of failure (SPOF): it holds the conversion keys, and is thus solely responsible for unlinkability and for controlling which data exchanges are permitted.

This paper introduces n-PEP, a distributed proxy re-encryption [4] scheme with built-in, transitive pseudonymization, that replaces this single converter with n semi-trusted *transcryptors*. It allows users[2] to share end-to-end encrypted data about data subjects without knowing the other user's pseudonym for that data subject. It does so asynchronously and non-interactively: data can be encrypted at its source and uploaded to a (central, e.g. cloud) intermediate storage without specifying the future recipient, and later be made decryptable for a recipient by *rekeying*, without the original sender being involved. Every transcryptor can independently monitor and control all exchanges, enforcing access rules and policies, while seeing only encrypted data. Following the principle of *distributed trust*, after a trusted initial setup that does not involve users, confidentiality and pseudonym unlinkability are maintained as long as at least one transcryptor remains uncompromised. Compartmentalization thus limits the impact of data leakage by (compromised) users by design, assuming the shared data itself is unlinkable (section 8).

As a (simplified) textbook example, three doctors (users) Alice, Bob and Charlie communicate about their patients (data subjects) via transcryptors Peter and Quinten ($n = 2$). Each doctor knows patient Xavier under a different pseudonym, e.g. Alice as `586176696572` and Bob as `52616e646f6d`, without knowing the others' pseudonyms. Only Peter and Quinten together can convert between them, blindly, without knowing Xavier at all. Each of them can independently monitor or block the exchange. Alice can also encrypt and store her patient records, perhaps at a central storage run by Sarah, without specifying a recipient, so Peter, Quinten and Sarah can later decide to send (parts of) them to Bob (or Charlie, or a new user), without Alice being involved.

Our work is based on the existing PEP (Polymorphic Encryption and Pseudonymization) framework [23], hereafter basic-PEP, an advanced method of pseudonymization [17] that exploits the malleable properties of ElGamal encryption [16]. While basic-PEP describes the underlying cryptographic primitives and sketches some potential use cases, a concrete system architecture has never been described in detail. This technology, however, has been implemented in several applications, including the *PEP Responsible Data Sharing Repository*[3] (hereafter PEP-repo) that is used for various long-term (medical) research

[1] Pseudonyms are also called context-specific or organization-specific [8]; we use domain-specific [6], in line with 2025 EU guidelines [15].

[2] In a broad sense, i.e. both natural persons and functional users for software systems.

[3] See `https://pep.cs.ru.nl` or `https://github.com/PEP-Repository/core`.

projects [18] and the Dutch national eID infrastructure [20,23]. Our experience implementing basic-PEP in such systems led to the identification of vulnerabilities and shortcomings that n-PEP resolves. In particular, these deployments feature mostly unidirectional data flows and originally relied on a single transcryptor, whereas n-PEP enables multidirectional data sharing and distributes transcryption.

This paper makes several contributions. First, we extend basic-PEP with reshuffling and rekeying operations *between* two domains, making pseudonymization reversible and transitive by construction and thereby enabling multidirectional data sharing (section 3). Second, we identify a *linking-oracle* vulnerability in the deployed PEP-repo, in which basic-PEP's *polymorphic pseudonyms* allow colluding users to link their pseudonyms (section 3.1), which n-PEP resolves by construction. Third, we distribute transcryption and session key establishment over n parties, without any persistently trusted party (section 4), so confidentiality and pseudonym unlinkability hold against $n-1$ compromised transcryptors, while every single transcryptor can independently monitor and block data sharing. Finally, we present a concrete system architecture with asynchronous data sharing (section 5), implemented in the open-source `libpep` software library, and analyze the security properties n-PEP achieves (section 6). We further describe applications (section 7), discuss the limits of data unlinkability (section 8), and compare n-PEP with basic-PEP and related work (section 9).

2 Background

We introduce the model of pseudonym systems (section 2.1) and the basic-PEP operations (section 2.2) on which n-PEP builds.

2.1 Pseudonym systems and pseudonymization

In a pseudonym system [7,22], a data subject is known to different organizations under different, unlinkable pseudonyms. Classically, data subjects interact with those organizations themselves, disclosing a pseudonymous credential to each. We consider a variant in which users (e.g. organizations or functional systems) instead communicate *about* data subjects, who take no part in the exchange. The pseudonym for data subject X in domain A, commonly denoted $X_{@A}$, can be converted into $X_{@B}$ by one or more central, semi-trusted parties, which we call *transcryptors*,[4] where $X_{@A}$ and $X_{@B}$ should be unlinkable for anyone but these (joint) parties. We associate each user with a different domain.[5] In the remainder of this paper, we consider data to consist of *attributes*, being unidentifiable and unlinkable facts about data subjects, and *pseudonyms*, as the only domain-specific identifiers for data subjects.

[4] Other common terms used for this are *transformer* [23] or *converter* [6,5].

[5] There may be use cases where multiple users share a domain, or where a single user is in multiple domains, but for simplicity we consider a 1–1 mapping in this paper.

Pseudonymization based on a randomized mapping table yields information-theoretic unlinkability, at a storage complexity of $\mathcal{O}(d)$ for d the size of the domain. Other methods, including ours, yield computational unlinkability at $\mathcal{O}(1)$ storage.

In particular, *blind* pseudonymization can be realized by oblivious evaluation of a pseudorandom function (PRF) such as the Diffie–Hellman PRF (DH-PRF), where the evaluating party applies the pseudonymization key without learning the input or output. In a two-party, *blinding-based* protocol, a requester masks the input, the evaluator applies the key, and the same requester unmasks the result, as in the standardized (V)OPRF protocol [11]. In a three-party, *encryption-based* protocol, the sender encrypts the input, the transcryptor applies the key homomorphically to the ciphertext, and the receiver decrypts the result, without the sender being involved. PEP follows this three-party approach.

Compartmentalization through pseudonymization inevitably requires the shared data itself, excluding the identifiers, to be unlinkable (i.e. not contain quasi-identifiers or other linkable attributes), which we assume for the remainder of this paper.

2.2 The basic-PEP framework

The PEP framework [23], hereafter basic-PEP, constructs a proxy re-encryption pseudonym system from three homomorphic operations on ElGamal ciphertexts [16], each applied blindly, i.e. without the performing party seeing the contents:

- **reshuffling** scales the plaintext by a scalar (M becomes $s \cdot M$), turning identifiers into pseudonyms that are unlinkable while s stays secret;
- **rekeying** makes a ciphertext decryptable by $k \cdot y$ instead of y without the performing party knowing y, a form of proxy re-encryption [4] enabling end-to-end encryption without involving the sender; and
- **rerandomization** re-encodes a ciphertext without changing its message or recipient, producing unlinkable copies.

Given a fixed generator point G on an elliptic curve, generating group $\mathbb{G}$, we have the following definitions:[6]

Definition 1 (ElGamal). *Let b be a non-zero random scalar. A plaintext message M can be encrypted for a public key Y as $\langle B, C\rangle$ and can be decrypted knowing secret key y (where $Y = y \cdot G$) as*

$$\mathsf{Enc}(b, M, Y) = \langle b \cdot G, M + b \cdot Y\rangle = \langle B, C\rangle$$
$$\mathsf{Dec}(\langle B, C\rangle, y) = C - y \cdot B = M$$

[6] Unlike basic-PEP, we use regular ElGamal $\langle B, C\rangle$ tuples instead of $\langle B, C, Y\rangle$ triples (and slightly different metavariables). The triple encoding makes the intended recipient explicit and can be used for debugging, but Y encodes no message-specific information and is not used for decryption (see Definition 1) and could thus be omitted, saving bandwidth and computations. Where Y is required (i.e. for rerandomization), it can be provided as an extra argument.

We thus consider encrypted messages to be ElGamal ciphertexts *of* plaintext message M, encrypted *for* (a user possessing) secret key y, being encrypted *with* associated public key Y. Message encoding into curve points is described in section A.

Definition 2 (Reshuffling). *Given a non-zero scalar $s \neq 1$, a ciphertext $\langle B, C\rangle$ of message M encrypted with Y can be transformed into a new ciphertext of message $s \cdot M$ as*

$$\mathsf{RS}(\langle B, C\rangle, s) = \langle s \cdot B, s \cdot C\rangle$$

such that $\mathsf{RS}(\mathsf{Enc}(b, M, Y), s) = \mathsf{Enc}(s \cdot b, s \cdot M, Y)$.

Reshuffling is the encryption-based oblivious evaluation of the DH-PRF $F_s(M) = s \cdot M$,[7] which is pseudorandom under the Decisional Diffie–Hellman (DDH) assumption when s is secret.

Definition 3 (Rekeying). *Given a non-zero scalar $k \neq 1$, a ciphertext $\langle B, C\rangle$ for secret key y can be transformed into a new ciphertext for $k \cdot y$ as*

$$\mathsf{RK}(\langle B, C\rangle, k) = \langle k^{-1} \cdot B, C\rangle$$

such that $\mathsf{RK}(\mathsf{Enc}(b, M, Y), k) = \mathsf{Enc}(k^{-1} \cdot b, M, k \cdot Y)$.

A transcryptor (e.g. Peter or Quinten) can use rekeying to make messages encrypted by Alice decryptable by a different user Bob, while reshuffling pseudonymizes any identifiers in the ciphertext from Alice's domain to Bob's domain. Neither operation reveals the message itself. The joint operations of either reshuffling and then rekeying of identifiers (which can be implemented with an integrated RSK operation), or only rekeying of attributes, are called **transcrypting**. The transcryptor can perform access control to determine whether transcryption between two users should be allowed. Throughout section 3 we consider a single transcryptor. In section 4 we extend to n transcryptors.

Confidentiality and unlinkability ultimately rely on the hardness of the discrete log problem; Curve25519 offers 128 bits of security [2].

A requirement that is not explicit in basic-PEP but is essential for security is that pseudonyms and attributes must be encrypted with different keys and rekeyed with different factors. Otherwise, encrypted pseudonyms could falsely be presented as encrypted attributes towards transcryptors, resulting in rekeying without reshuffling, leaking another user's pseudonym value.

Definition 4 (Rerandomization). *Given a non-zero scalar r, a ciphertext $\langle B, C\rangle$ encrypted with Y can be transformed into a new ciphertext of the same message and for the same key as*

$$\mathsf{RR}(\langle B, C\rangle, Y, r) = \langle r \cdot G + B, r \cdot Y + C\rangle$$

such that $\mathsf{RR}(\mathsf{Enc}(b, M, Y), Y, r) = \mathsf{Enc}(b + r, M, Y)$.

[7] Strictly, the DH-PRF is $F_s(M) = s \cdot H(M)$ for a hash H into the group [11]: without H, F_s is linear and thus not pseudorandom for chosen inputs. We omit H since identifiers are either randomly selected group elements or hashed into the group upon encoding (section A).

Rerandomization can be used to create unlinkable copies of the same (stored) ciphertext. It additionally prevents a *plaintext injection attack* at transcryption: a malicious sender could place a plaintext pseudonym M directly in C, so that an RSK operation returns the reshuffled pseudonym $s \cdot M$ in the clear, bypassing rekeying. Mixing fresh randomness into the ciphertext before reshuffling (an integrated RRSK or RRK operation) keeps $s \cdot M$ hidden even on malformed inputs.

3 Multidirectional data sharing

Classical pseudonymization goes in a single direction, transforming an identifier (or *nym*) into an unlinkable pseudonym in a different domain. Reversing this transformation typically requires a special recovery operation (such as a reverse lookup in a mapping table), and converting between two pseudonyms in different domains then means reversing first and pseudonymizing again, potentially allowing for linkage in-between.

Basic-PEP also mainly considers pseudonymization in a single direction. Our experience implementing basic-PEP in applications with multidirectional data sharing led us to a slightly more generic model, where pseudonyms can be converted between domains back and forth. For simplicity, we also call the original identifier a pseudonym, in its own origin domain. For this model, we identify two additional properties for pseudonymization to be applied securely and efficiently:

- **Reversibility**: pseudonymizing from domain A to domain B and back to A should yield the same pseudonym as in domain A initially.
- **Transitivity**: pseudonymizing from domain A to domain B, and then to domain C, should yield the same pseudonym as pseudonymizing from A to C directly.

Basic-PEP does not satisfy these properties *by construction*. Reshuffling as defined in Definition 2 transforms an identifier M into a (computationally) unlinkable pseudonym $M' = s \cdot M$. With s being the result of a simple hash function as proposed in basic-PEP (i.e. $s = \mathsf{hash}(u)$ with u the receiving user's domain), pseudonymization is not reversible: if Alice sends data about data subject X to Bob, Bob is not able to send data back to Alice about the same X, as Alice would receive a different pseudonym back (since $\mathsf{hash}(A) \neq \mathsf{hash}(B)^{-1}$). And when a third user Charlie joins the conversation, they would receive a different pseudonym for X from Bob than from Alice. While basic-PEP describes a workaround for the reversibility problem, we show in section 3.1 that this workaround acts as a linking oracle, breaking pseudonym unlinkability.

To satisfy the desired properties, we introduce an extended reshuffling operation that is both invertible (reversible) and transitive by construction, using both the factor for the sending user, s_{from}, and that for the receiving user, s_{to}, instead of the receiving user only.

Definition 5 (Two-domain reshuffling). *Given two non-zero scalars $s_{from}, s_{to} \neq 1$, a ciphertext $\langle B, C \rangle$ of $s_{from} \cdot M$ can be transformed into a new ciphertext of*

$s_{to} \cdot M$ *as:*

$$\mathsf{RS2}(\langle B, C\rangle, s_{from}, s_{to}) = \langle s_{from}^{-1} \cdot s_{to} \cdot B, s_{from}^{-1} \cdot s_{to} \cdot C\rangle$$

Lemma 1 (Transitivity and invertibility). *For any ciphertext V and non-zero scalars s_A, s_B, s_C: $\mathsf{RS2}(\mathsf{RS2}(V, s_A, s_B), s_B, s_C) = \mathsf{RS2}(V, s_A, s_C)$, since the s_B factors cancel. Invertibility is the special case where $s_C = s_A$.*

We thus effectively apply the original RS operation with $s = s_{\text{from}}^{-1} \cdot s_{\text{to}}$, pseudonymizing from a local pseudonym to another local pseudonym, instead of from a global *polymorphic pseudonym* as in basic-PEP. A similar construction can be used for rekeying between two session keys ($\mathsf{RK2}$, or RK with $k = k_{\text{from}}^{-1} \cdot k_{\text{to}}$) instead of a global key and a session key as in basic-PEP, as is also done for bidirectional proxy re-encryption schemes without pseudonymization [4], which we omit for readability.

The significance of $\mathsf{RS2}$ lies not in its algebra, which is deliberately minimal, but in the model shift it enables: all pseudonyms become local and conversions become symmetric between domains. This removes the need for the global polymorphic pseudonyms whose insecurity we demonstrate next.

3.1 Polymorphic pseudonyms as a linking oracle

Basic-PEP distinguishes encrypted pseudonyms before and after reshuffling. Encrypted *local pseudonyms*[8] are generated by reshuffling an encrypted *global* pseudonym, called the *polymorphic pseudonym*.[9] There is no direct way to go from one local pseudonym to another, or back to the polymorphic pseudonym. When this is required, transcryptors instead send a rerandomized copy of the polymorphic pseudonym along with the (reshuffled and rekeyed) encrypted local pseudonym, which users can use to send data back.

This construction breaks pseudonym unlinkability. The fundamental problem is that a polymorphic pseudonym is not bound to any recipient's domain: it is valid across all users. Users cannot see its contents (it is encrypted for a key they do not possess) nor link copies directly (they are rerandomized every time), but they *can* exchange polymorphic pseudonyms among each other and use them. This gives a *linking oracle* whenever a polymorphic pseudonym can trigger an action whose effect is observable under a user's own local pseudonym. A user injects a polymorphic pseudonym obtained from another user, observes the effect under their own local pseudonym, and thereby links the two local pseudonyms, without the users communicating via the system at all.

We identified an instance of this vulnerability in PEP-repo. There, data is mainly shared in one direction, from initial collection to an analyst, but some users that receive data also need to send data back; for example, a doctor may

[8] Sometimes also called *persistent pseudonyms*.

[9] This terminology can be confusing, since all encrypted pseudonyms are polymorphic, because they can be rerandomized.

receive an MRI scan from the repository and (re-)upload a diagnosis. They receive a rerandomization of the polymorphic pseudonym to do this. Bob, however, can use the polymorphic pseudonym that Alice received to upload data back. Because PEP-repo allows users to see the status of an upload, Bob then observes under which of his own local pseudonyms the data was uploaded, and thereby learns that his pseudonym and Alice's refer to the same data subject.

This is not a traffic or timing analysis attack: only Bob (not Alice!) communicates with the repository, and the transcryptors do not observe any communication between Alice and Bob.

In contrast, in n-PEP, all pseudonyms are local and generated by reshuffling *from* one domain *to* another,[10] so users only receive domain-specific pseudonyms. Additionally, sending polymorphic pseudonyms alongside local pseudonyms requires more bandwidth and additional rerandomization, so eliminating them is also more efficient.

4 Eliminating single points of failure

The primitives from section 2.2 and 3 enable blind pseudonymization and rekeying of encrypted messages between users. This prevents transcryption from being a privacy hotspot. But when performed by a single party, this party is still a SPOF for pseudonym unlinkability: a single transcryptor knows the secret reshuffling factors it applies and thus how unencrypted pseudonyms relate, even though during communication it only sees encrypted messages. When these secrets leak, pseudonyms can be linked across all domains.

To solve this, we distribute transcryption over n parties, each applying their own secret factors during transcryption. The ultimate rekeying and reshuffling factors then consist of the product of every party's individual secret factors, i.e. $k = \prod_{i=1}^{n} k_i$ and $s = \prod_{i=1}^{n} s_i$ (both non-zero and $\neq 1$ except with negligible probability). This way, secrecy of k and s, and thus unlinkability and confidentiality, are maintained as long as at least one party remains uncompromised.[11]

We actually require domain-specific reshuffling factors s_{d_i} and session-specific rekeying factors k_{c_i} for domain d and session c. These *factors* can be deterministically derived from fixed party-specific *secrets* S_i and K_i using a hash function[12] as $s_{d_i} = \mathsf{hash}(S_i \,|\, d)$ and $k_{c_i} = \mathsf{hash}(K_i \,|\, c)$. For readability, however, we just use s_i and k_i in the remainder of this paper.

Definition 6 (Distributed transcryption). *Let k_i and s_i be (pseudo)random secret rekeying and pseudonymization factors from party i. In an n-PEP system,*

[10] The global pseudonym for local pseudonym M would be of the value $s_{\text{from}}^{-1} \cdot M$ for associated s_{from}, but should never exist in that form.

[11] Distribution does cost availability, since all n transcryptors must be online; standard threshold techniques relax this to t-of-n, but then t transcryptors suffice to link pseudonyms or authorize exchanges.

[12] Depending on the implementation an HMAC construction might be required to prevent length extension attacks on d and c.

ciphertexts for y are rekeyed for $k \cdot y$ with rekeying factor $k = \prod_{i=1}^{n} k_i$, and reshuffled with pseudonymization factor $s = \prod_{i=1}^{n} s_i$.[13]

$$\mathsf{RK}(\mathsf{RK}(V, k_1), k_2) = \mathsf{RK}(V, k_1 \cdot k_2) = \mathsf{RK}(V, k)$$
$$\mathsf{RS}(\mathsf{RS}(V, s_1), s_2) = \mathsf{RS}(V, s_1 \cdot s_2) = \mathsf{RS}(V, s)$$

Lemma 2 (Commutativity and associativity). *By commutativity and associativity of scalar multiplication and Definition 6, reshuffling and rekeying are commutative and associative operations with regard to a message's plaintext and the key it is encrypted for.*

Rerandomization distributes trivially, with factors composing additively ($r = \sum_{i=1}^{n} r_i$). The protection against plaintext injection (section 2.2) thereby holds as long as at least one transcryptor rerandomizes honestly.

4.1 Session keys and system setup

As mentioned in basic-PEP, there must be a *separation of duties* between parties that perform transcryption and parties that provide users with their (secret) session keys,[14] as a single party could otherwise decrypt all messages it sees during transcryption with the decryption keys it provides. By distributing transcryption, we can also securely implement this key establishment, distributed over the same transcryptors, using a simple multi-party computation.

System setup During system setup, a trusted authority generates a random global secret key y, turns it into a blinded global secret key y'_b using n blinding factors b_i, one from each transcryptor, as per Definition 7, and discards y afterward. The trusted authority is a single point of failure, but it exists only for setup: it discards y and can then be decommissioned entirely. Crucially, it never sees any transcryptor's rekeying and pseudonymization secrets S_i and K_i, so even a compromised setup authority cannot link pseudonyms.

Definition 7 (System setup). *During system setup, a trusted authority generates a random global secret key y for the system and computes the blinded global key y'_b using random blinding factors of all n transcryptors in the system as:*

$$y'_b = y \cdot \prod_{i=1}^{n} b_i^{-1}$$

Session key establishment After setup, users can securely establish a session key $y_k = k \cdot y$ (without revealing k or y to anyone) by acquiring n session key shares $u_i = b_i \cdot k_i$ from each transcryptor i (after authentication[15]). These session key shares *blind* the actual value k_i using a transcryptor's secret blinding factor b_i and distribute the global secret key.

[13] Similar for RS2, RK2, RRK, RRK2, RSK, RSK2, RRSK and RRSK2.
[14] This separation alone does not resolve the aforementioned SPOF.
[15] The exact method of authentication we consider out-of-scope.

Definition 8 (Session key establishment). *Secret session key y_k (for associated public $Y_k = k \cdot Y = k \cdot y \cdot G = y_k \cdot G$) is constructed from blinded global key $y'_b = y \cdot \prod_{i=1}^{n} b_i^{-1}$ and n session key shares $u_i = b_i \cdot k_i$ from each transcryptor as:*

$$y_k = y'_b \cdot \prod_{i=1}^{n} u_i = y \cdot \prod_{i=1}^{n} b_i^{-1} \cdot \prod_{i=1}^{n} b_i \cdot k_i = k \cdot y$$

Users can maintain different sessions with each transcryptor, and should update their session key with the new session key share whenever they refresh their session with any transcryptor.

5 *n*-PEP system architecture

We sketch the following interaction between users and transcryptors using n-PEP, assuming users to be associated with their own domain:

1. A sender establishes (secret) session key $k_{\text{from}} \cdot y$ after authentication towards all n transcryptors (Definition 8).
2. The sender encrypts message M using its own public key $k_{\text{from}} \cdot y \cdot G$ into ciphertext $V_{k_{\text{from}} \cdot Y}$.
3. The recipient retrieves (secret) session key $k_{\text{to}} \cdot y$ after authentication towards all n transcryptors (Definition 8).
4. The sender sends the ciphertext $V_{k_{\text{from}} \cdot Y}$ to the recipient through every transcryptor in the system. Each transcryptor performs an RRSK2 operation on pseudonyms, or an RRK2 on attributes, with its own factors, transcrypting the ciphertext in n steps into $V'_{k_{\text{to}} \cdot Y}$.
5. The recipient decrypts $V'_{k_{\text{to}} \cdot Y}$ into M' using this secret session key. For attributes, this will be equal to the original attributes ($M' = M$). For identifiers, this will be a (pseudo)randomly reshuffled identifier or recipient-specific pseudonym ($M' = s \cdot M$, with s the combined factor of Definition 6).

Each transcryptor can hereby monitor all requests and perform access control: a transcryption request reveals the authenticated requesting user and the source and target pseudonymization domains, but not the pseudonym values or data contents. On this basis, every transcryptor independently decides whether the requested conversion is permitted, for example against a policy of allowed sender–recipient pairs. Because all n transcryptors must cooperate for any transcryption, any single transcryptor can block unauthorized pseudonym conversion or data sharing, while no transcryptor alone can enable it.

Unlike basic-PEP, messages are encrypted with a sender-specific session key $k_{\text{from}} \cdot Y$ instead of the global public key Y, symmetrical with how pseudonyms are computed (section 3). This also enables the sender to prove message authenticity, which we develop in ongoing work (section 10). The global-key approach remains usable with minimal changes when the sender cannot interactively obtain a session key (e.g. the encryption key is hardcoded in hardware or the sender is offline) or when session keys would act as a quasi-identifier and introduce linkability (section 8).

5.1 Asynchronous data sharing

The sender encrypts its messages in step 2 *before* the recipient establishes a session key and *without* specifying the future recipient. The sender's encrypted ciphertexts can thus be stored at an intermediate storage (e.g. in the cloud, together with non-confidential metadata describing them) that cannot decrypt them, and be shared multiple times with different recipients at any time in the future, without the original sender being involved. We thus consider a slightly more advanced system model, optionally including intermediate storages. When messages are uploaded once and shared multiple times, the intermediate storage should rerandomize the ciphertexts to prevent linkability of the stored ciphertexts against the transcryptors or anyone observing the network.

In contrast, regular end-to-end encrypted data sharing requires the sender to encrypt messages for each recipient separately.[16] Our approach enables special use cases, such as a data repository, where data is shared over long periods of time and data management takes place on encrypted data, by authorities that cannot decrypt the data itself, without the original sender being involved.

5.2 Implementation

We implemented (n-)PEP in the open-source `libpep` software library,[17] written in Rust with Python and WebAssembly bindings, using the `curve25519-dalek` crypto-library. The computational cost is dominated by elliptic curve scalar multiplications: per transcryptor, at most three per RSK2 per message (four for an integrated RRSK2) and three per session, achieving about three pseudonym conversions per millisecond, single-threaded, on commodity hardware. Since messages are independent, computation scales linearly with the number of messages and transcryptors.

6 Security properties

In this section, we discuss the security properties n-PEP achieves and sketch why they hold. Our system model consists of users (senders and recipients) communicating with each other through n central transcryptors, with optional intermediate storages as discussed in section 5.1. We assume an attacker who compromises (i.e. learns all secrets of) at most $n-1$ transcryptors and any number of intermediate storages or other users not involved in the communication. We further assume secure, authenticated connections between all users, transcryptors and storages, a secure initial setup of transcryptors as described in section 4.1, and the DDH assumption in the Ristretto255 group. Finally, we inevitably assume data unlinkability (section 2.1) and the absence of other linkage channels such as traffic or timing analysis; we discuss both in section 8.

[16] In reality, one would apply box encryption, encrypting messages symmetrically and then encrypting the symmetric key for each recipient separately asymmetrically. This, however, has consequences for linkability, as we discuss in section A.2.

[17] See `https://github.com/NOLAI/libpep` and `https://crates.io/crates/libpep`.

Our scheme has the following security properties:

1. **Attribute confidentiality**: the value of attributes sent between sender and recipient remains confidential to anyone but the sender and recipient.
2. **Pseudonym confidentiality**: the value of a user's pseudonym for a specific data subject remains confidential against any other user, including compromised recipients.
3. **Pseudonym unlinkability**: given two pseudonyms in two different pseudonymization domains (i.e. from two different users), it is not possible to tell whether they are related (i.e. for the same data subject) without at least one uncompromised transcryptor cooperating.[18] This property also holds against an attacker model including both compromised senders and compromised recipients: even two compromised users cannot link their pseudonyms (if the attributes are unlinkable).
4. **Cross-encryption ciphertext unlinkability**: given two ciphertexts from different encryptions, it is not possible to tell whether they encrypt the same plaintext. This also implies that an adversary cannot test whether a given ciphertext encrypts a known plaintext, such as their own pseudonym, since they could construct such a ciphertext themselves.
5. **Cross-rerandomization ciphertext unlinkability**: given two ciphertexts, it is not possible to tell whether one is a rerandomized copy of the other (i.e. whether they originate from the same stored ciphertext). This property holds against a slightly weaker attacker model, excluding compromised storages involved in the communication.

These properties follow from the structure of the scheme. Decrypting any (intermediate) ciphertext requires a session key, which by Definition 8 requires all n session key shares, of which at least one remains unknown. Linking pseudonyms $s_A \cdot M$ and $s_B \cdot M$ across domains requires the ratio s_A/s_B, which is also a product over all transcryptors' secret factors of which at least one is unknown, and computing it from the pseudonyms alone contradicts the DDH assumption; reversibility and transitivity (section 3) do not weaken this, being equally conditional on these factors. Transitivity is thus not the vulnerability it is in traditional proxy re-encryption (see also section 9). Cross-encryption unlinkability follows from the semantic security of ElGamal under DDH. Cross-rerandomization unlinkability follows from rerandomized copies being distributed identically to fresh encryptions. Since every transcryptor rerandomizes, this also makes the intermediate ciphertexts at different hops unlinkable.

While basic-PEP aims to achieve the same security properties with a single trusted transcryptor, its polymorphic pseudonyms act as a linking oracle in practice (section 3.1). In n-PEP, this oracle is absent by construction: users never hold polymorphic pseudonyms, and every conversion, including reversal, passes through all n transcryptors' access control (section 5).

[18] Pseudonym unlinkability implies pseudonym confidentiality.

7 Applications

We discuss applications of n-PEP: two existing deployments of basic-PEP that it improves, and new use cases it enables, all relying on pseudonym unlinkability to prevent identification of data subjects in specific contexts.

Notably, a 2025 judgment of the Court of Justice of the European Union clarified that pseudonymized data can be considered non-personal data from the perspective of a data controller that lacks the means to re-identify data subjects [10]. n-PEP's distributed architecture could realize exactly this condition, depending on the legal roles of the users and the parties hosting transcryptors, significantly reducing compliance burden under the GDPR.

Research data repository In PEP-repo, used in research projects such as the large-scale *Personalized Parkinson Project* [18], participant data is encrypted right at the source (e.g. a heart-rate measuring smartwatch) and stored centrally. Researchers, doctors or labs then get access to their own pseudonymized subsets of data, with transcryptors checking specific access rules before deciding to transcrypt, possibly years after collection. Distributed transcryption (section 4) is already implemented in PEP-repo; n-PEP additionally resolves its linking-oracle vulnerability (section 3.1).

National eID systems The Dutch national eID system uses basic-PEP for pseudonymization of citizen service numbers (BSN), in its *BSNk PP* infrastructure [21,20], used among others by the *DigiD Hoog* authentication service. Here, data is stored decentrally on eID smartcards, and transcryption is performed by a single trusted server, deployed in dedicated Hardware Security Modules (HSMs) with a replica at each authentication provider. While this server is not a privacy hotspot (it sees only encrypted messages), it is a SPOF [21]: replication mitigates the SPOF for availability, but every replica holds the same conversion key and authorizes conversions on its own, so the SPOF for pseudonym unlinkability remains. n-PEP could split this key over independent organizations, each possibly still using HSMs, so that any conversion requires them all.

Moreover, pseudonymization currently works unidirectionally, from eID smartcard via authentication provider to service provider: each organization receives its own pseudonym, but pseudonyms of different organizations cannot be converted into each other. Organizations that want to share data about the same citizen must still use shared identifiers. n-PEP would enable such data exchange without shared identifiers.

Multi-party data sharing More generally, transitive pseudonymization enables controlled multidirectional data sharing in *complex data processing systems* [14], where multiple parties process personal data about common data subjects and their exchanges must be monitored or restricted.[19] Examples include the Dutch

[19] When data sharing happens synchronously, one may decide to use n-PEP only for pseudonymization and share attributes via other end-to-end secure encryption schemes.

education sector, where schools and their software service providers could process student data under unlinkable pseudonyms instead of the currently shared ECK-iD[20], but also healthcare, government agencies [5], financial crime detection, and upcoming (European) data spaces.[21]

8 Data (un)linkability

The properties of section 6 concern the pseudonyms themselves; in practice, linkability may still arise in other ways, such as through the shared data itself, protocol metadata, or cryptographic artifacts of an implementation. We discuss these below.

Linkable data Pseudonym unlinkability inevitably only holds if the subsets of data shared between users contain no linkable attributes or quasi-identifiers (i.e. combinations of attributes that are unique and thus linkable), as assumed in section 2.1. While this may not be realistic for every application, the condition does hold in many use cases with low-entropy data, and applying pseudonymization at fine granularity can eliminate quasi-identifiers [13].

Protocol metadata So far, we considered traffic and timing analysis out-of-scope, but in real-time applications, two compromised users could trivially link pseudonyms through them. One mitigation is batch transcryption: transcrypting multiple encrypted pseudonyms and attributes in a single request, with transcryptors randomly permuting their order in the batch to prevent linkability based on batch position.

Cryptographic artifacts Finally, implementations may accidentally introduce linkable cryptographic artifacts. For example, each transcryptor requires the session name of the original encryption to derive the correct rekeying factors, so in asynchronous communication, the intermediate storage must store these session names alongside the ciphertext (n times, one for each transcryptor). Depending on the use case, these can be linkable, for example when data for only one data subject is processed per session. To prevent this, either initial encryption should use global keys as in basic-PEP, or upon storage, data should first be transcrypted towards a global encryption context (with $k_{\text{to}} = 1$ and thus $k = k_{\text{from}}^{-1}$) and later be transcrypted from this global context to the recipient's session. In section A.2, we show another instance: how box encryption in PEP-repo introduces linkable ciphertexts, and how this can be handled.

9 Related work

Our system model is closest to that of Camenisch and Lehmann (the CL system) [5], describing how distributed (governmental) databases can use unlinkable

[20] See `https://www.eck-id.nl`.
[21] See `https://digital-strategy.ec.europa.eu/en/policies/data-spaces`.

pseudonyms, with *servers* communicating about *users* via a single trusted *converter* that transitively converts pseudonyms. Cryptographically, however, the CL system derives pseudonyms with the DY-PRF [12] and makes generation and conversion verifiable using structure-preserving signatures, which require a bilinear group and cost both parties tens of exponentiations and pairings per conversion. (n-)PEP instead converts with plain scalar multiplications on standard curves such as Curve25519 (section 2.2), at three per transcryptor, but offers no such verifiability. Regardless, the CL converter remains a SPOF for pseudonym unlinkability, knowing all secret conversion factors, whereas n-PEP remains secure while $n-1$ transcryptors are compromised.

ScrambleDB [19] replaces the DY-PRF of the CL system with the DH-PRF over ElGamal encryption, the same algebraic core as (n-)PEP, formalized as an oblivious and convertible PRF (coPRF). It is thus cryptographically the closest system to ours, but still considers a single converter, which is a SPOF for unlinkability, whereas n-PEP distributes this trust and additionally supports asynchronous use cases. Moreover, while the CL system and ScrambleDB are research prototypes, (basic-)PEP is deployed in production systems (section 7).

Finally, the standardized OPRF protocol [11] achieves blind evaluation of the same DH-PRF in the two-party setting, where the same party blinds the input and unblinds the output, in contrast to n-PEP's three-party setting with distinct senders, transcryptors and recipients (section 2.1). Its VOPRF mode makes evaluation verifiable using lightweight Schnorr-style zero-knowledge proofs; bringing such proofs to the three-party setting is the verifiability we consider future work for n-PEP (section 10). Table 1 summarizes the differences.

Table 1: Feature comparison with related systems, by setting (section 2.1).

	2-party	3-party			
Feature	(V)OPRF	CL	ScrambleDB	basic-PEP	n-PEP
Transitive pseudonymization (section 3)	×	✓	✓	○[1]	✓
Distributed transcryption (section 4)	×	×	×	×	✓
Asynchronous use cases	×	×	×	✓	✓
Pairing-free, standard curves	✓	×	✓	✓	✓
Verifiable pseudonymization	✓	✓	×	×	×[2]

[1] Using insecure polymorphic pseudonyms.
[2] Future work with promising results.

Proxy re-encryption More broadly, n-PEP can be seen as a bidirectional, multi-hop proxy re-encryption (PRE) scheme [4] with built-in pseudonymization. Traditional PRE schemes instead treat these properties as vulnerabilities [4,1]: transitivity lets a single proxy derive re-encryption keys that were never authorized, and bidirectionality lets it collude with a recipient to recover the sender's key. Combining multi-hop conversion with unidirectionality and chosen-ciphertext

security was left as an open problem [9]. In our model, the transcryptors are themselves the authorization point for conversions, making transitivity a feature, while distribution ensures no individual party can transform or decrypt data without cooperation from all others.

10 Future work

While n-PEP guarantees confidentiality and unlinkability, it does not guarantee integrity or authenticity of pseudonyms or attributes: transcryptors could inconsistently transcrypt messages, resulting in corrupted pseudonyms, and intermediate storages could send unauthentic or corrupted data. Ongoing research by the authors shows promising results using zero-knowledge proofs to enforce pseudonym consistency and message integrity throughout the transcryption process, with the sender's session key additionally proving message authenticity, extending n-PEP to fully malicious transcryptors and storages.

Furthermore, the security of (n-)PEP ultimately rests on the hardness of the discrete logarithm problem; quantum-safe PEP primitives, possibly based on RLWE cryptography, are required for long-term security, which is especially relevant in the repository use case where data may be stored for a long time.

11 Conclusion

We presented n-PEP: an architecture for secure, asynchronous data sharing with distributed pseudonymization, which can be considered a distributed proxy re-encryption scheme with built-in pseudonymization, implemented in the open-source `libpep` software library. n-PEP extends basic-PEP with reversible and transitive operations (section 3), enabling multidirectional data sharing and eliminating the *polymorphic pseudonyms* that we have shown to break pseudonym unlinkability. Moreover, n-PEP distributes transcryption over n parties (section 4), following the principle of *distributed trust*: confidentiality and pseudonym unlinkability hold against $n-1$ compromised transcryptors (section 6), eliminating single points of failure. Because pseudonyms can only be converted by the transcryptors, every transcryptor can independently monitor all data sharing and can block unauthorized exchanges to enforce specific data sharing policies, even with colluding users, while no transcryptor alone can read data or link pseudonyms. n-PEP improves two existing basic-PEP deployments (section 7): the PEP-repo research data repository and the Dutch national eID system. Its transitive pseudonymization also unlocks new use cases for multidirectional data sharing, without a single trusted converter, potentially even rendering the shared data non-personal under the GDPR (section 7).

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

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

References

1. Ateniese, G., Fu, K., Green, M., Hohenberger, S.: Improved proxy re-encryption schemes with applications to secure distributed storage. ACM Transactions on Information and System Security **9**(1), 1–30 (2006). https://doi.org/10.1145/1127345.1127346
2. Bernstein, D.J.: Curve25519: New Diffie-Hellman speed records. In: Public Key Cryptography – PKC 2006. pp. 207–228 (2006). https://doi.org/10.1007/11745853_14
3. Bernstein, D.J., Hamburg, M., Krasnova, A., Lange, T.: Elligator: Elliptic-curve points indistinguishable from uniform random strings. In: ACM Conference on Computer and Communications Security – CCS 2013. pp. 967–980 (2013). https://doi.org/10.1145/2508859.2516734
4. Blaze, M., Bleumer, G., Strauss, M.: Divertible protocols and atomic proxy cryptography. In: Advances in Cryptology – EUROCRYPT '98. pp. 127–144 (1998). https://doi.org/10.1007/BFb0054122
5. Camenisch, J., Lehmann, A.: (Un)linkable pseudonyms for governmental databases. In: ACM Conference on Computer and Communications Security – CCS 2015. pp. 1467–1479 (2015). https://doi.org/10.1145/2810103.2813658
6. Camenisch, J., Lehmann, A.: Privacy-preserving user-auditable pseudonym systems. In: IEEE European Symposium on Security and Privacy – EuroS&P 2017. pp. 269–284 (2017). https://doi.org/10.1109/EuroSP.2017.36
7. Chaum, D.: Security without identification: Transaction systems to make big brother obsolete. Communications of the ACM **28**(10), 1030–1044 (1985). https://doi.org/10.1145/4372.4373
8. Chaum, D.: Showing credentials without identification. In: Advances in Cryptology – EUROCRYPT '85. pp. 241–244 (1986). https://doi.org/10.1007/3-540-39805-8_28
9. Chow, S.S.M., Weng, J., Yang, Y., Deng, R.H.: Efficient unidirectional proxy re-encryption. In: Progress in Cryptology – AFRICACRYPT 2010. pp. 316–332 (2010). https://doi.org/10.1007/978-3-642-12678-9_19
10. CJEU: Judgment in case C-413/23 P (SRB v EDPS) (2025), https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=ecli:ECLI:EU:C:2025:645
11. Davidson, A., Faz-Hernandez, A., Sullivan, N., Wood, C.A.: Oblivious pseudorandom functions (OPRFs) using prime-order groups. RFC 9497 (2024). https://doi.org/10.17487/RFC9497
12. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: Public Key Cryptography – PKC 2005. pp. 416–431 (2005). https://doi.org/10.1007/978-3-540-30580-4_28
13. Doesburg, J., van Gastel, B., Poll, E.: Pseudonymization as a service: Privacy-preserving system evolution through micropseudonymization. In: Belgium-Netherlands Software Evolution Workshop – BENEVOL 2025. CEUR-WS (2025), https://benevol2025.github.io/pre/paper12.pdf, to appear
14. Doesburg, J., van Gastel, P., van Gastel, B., Poll, E.: Data processing diagrams – a modeling technique for privacy in complex data processing systems. In: Privacy and Identity Management – Privacy and Identity 2024. pp. 115–131 (2025). https://doi.org/10.1007/978-3-031-91054-8_6
15. EDPB: Guidelines 01/2025 on pseudonymisation (2025), https://www.edpb.europa.eu/our-work-tools/documents/public-consultations/2025/guidelines-012025-pseudonymisation_en

16. Elgamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory **31**(4), 469–472 (1985). https://doi.org/10.1109/TIT.1985.1057074
17. ENISA: Pseudonymisation techniques and best practices: Recommendations on shaping technology according to data protection and privacy provisions (2019). https://doi.org/10.2824/247711
18. van Gastel, B.E., Jacobs, B., Popma, J.: Data protection using polymorphic pseudonymisation in a large-scale Parkinson's disease study. Journal of Parkinson's Disease **11**(s1), S19–S25 (2021). https://doi.org/10.3233/JPD-202431
19. Lehmann, A.: ScrambleDB: Oblivious (chameleon) pseudonymization-as-a-service. Proceedings on Privacy Enhancing Technologies **2019**(3), 289–309 (2019). https://doi.org/10.2478/popets-2019-0048
20. Logius: Privacy impact assessment DigiD Hoog (2017), https://www.eerstekamer.nl/overig/20190129/privacy_impact_assessments_pia
21. Logius: BSNk PP technische specificaties v11.1 (2024), https://gitlab.com/logius/bsnk/bsnk-techspecs/bsnk/-/raw/main/BPTSlive.pdf
22. Lysyanskaya, A., Rivest, R.L., Sahai, A., Wolf, S.: Pseudonym systems. In: Selected Areas in Cryptography – SAC 1999. pp. 184–199 (2000). https://doi.org/10.1007/3-540-46513-8_14
23. Verheul, E., Jacobs, B.: Polymorphic encryption and pseudonymisation in identity management and medical research. Nieuw Archief voor Wiskunde **18**(3), 168–172 (2017)
24. Westerbaan, A., Hendriks, L.: Polymorphic encryption and pseudonymisation of IP network flows. In: IFIP Networking Conference – Networking 2020. pp. 494–498 (2020), https://ieeexplore.ieee.org/document/9142777

A ElGamal message encoding

To share pseudonyms or attributes (hereafter messages) via PEP, they must be encoded as a group element of $\mathbb{G}$. We use Curve25519 [2] with Ristretto,[22] which eliminates the complications of points with small-order factors (cofactors) so that any point can be used safely. This gives a message space of $l = 2^{252} + 27742317777372353535851937790883648493$, with points represented as 32 bytes. When an attribute is too large to fit a single group element, it can be split over multiple messages.

A.1 Lizard encoding

Although Ristretto points can be canonically encoded with 32 bytes, not all 32-byte values (only $\frac{l}{2^{256}} \approx 6.25\%$) are valid Ristretto points. For pseudonyms, this does not necessarily cause problems as they can be randomly selected from valid points. For attributes, or when using existing values (such as email addresses) as origin for pseudonyms, however, this is problematic. While any data can be hashed to a valid point using the Elligator map [3], this is not a bijective mapping, making it impossible to encode and later decode to the same data.

[22] See https://ristretto.group.

As a solution, we use the *lizard* method, also discussed by Westerbaan and Hendriks [24]. It combines the *elligator2* map [3] with hashing to map arbitrary 16-byte values bijectively to (32-byte) Ristretto points and back, with extremely high probability.[23] This way, (practically) all 16-byte values can be encoded and decoded back into the same 16-byte value.[24]

Lizard encoding for pseudonyms While any 16-byte value can be mapped to a valid 32-byte Ristretto point and back using lizard, the reverse is not true: not all valid Ristretto points are valid lizard encodings. When using lizard for encoding an existing (16-byte) identifier as origin pseudonym, a consequence is that after reshuffling, the newly derived pseudonyms are highly unlikely to be valid lizard points and cannot be represented with 16 bytes. The reshuffled pseudonym will be a pseudo-random Ristretto point that likely does not have a lizard encoding, since reshuffling takes place in the 32-byte Ristretto space and not in the 16-byte lizard space. Reshuffled pseudonyms must thus always be represented as 32-byte values, only the origin pseudonym can be represented as the original 16-byte (lizard) value.

A.2 Symmetric (box) encryption

As discussed, whenever an attribute's size exceeds the available message space (i.e. 16 bytes for arbitrary data, or 15 bytes including padding, using Curve25519 with Ristretto), attributes can be split over multiple messages. For efficiency, however, one could alternatively encrypt larger attributes symmetrically (for example with AES), using a randomly generated key that fits the available message size and only share this key via PEP. The symmetrically encrypted attributes can then be shared via a different channel and decrypted using this key. This practice is also known as *box encryption*. This is significantly more efficient than n-PEP, because of hardware support for AES operations.

PEP-repo implements this **for all attributes, regardless of size**. It mainly does so for efficiency and to simplify implementation: PEP-repo supports both large files, such as MRI scans, which take multiple megabytes and are slow to encrypt asymmetrically, and small attributes of a couple of bytes. Another reason for this choice, however, is to avoid lizard encoding, which was unavailable during first implementation.

This approach with box encryption, however, violates data unlinkability in use cases with asynchronous data sharing: the symmetric ciphertext and key are unique, and thereby themselves become linkable data.

With typical symmetric encryption schemes, ciphertexts cannot be rerandomized as we do require for ciphertexts of n-PEP. Without this rerandomization, we lose the polymorphic properties that are required for cross-rerandomization

[23] i.e. with a probability of 99.9%, there are fewer than 80 bad inputs, or a chance of 1 in 2^{122} of hitting an invalid input, see `https://github.com/bwesterb/go-ristretto`.

[24] In order to support arbitrary length data, a padding scheme such as PKCS7 needs to be applied, sacrificing one byte.

ciphertext unlinkability. By comparing the symmetric key or the symmetric ciphertext, two users can thus observe whether they received copies of the same data for the same subject, and consequently link each other's pseudonyms, even though the plaintext data itself is indistinguishable.

A.3 High-entropy and low-entropy data

The introduction of linkability when applying box encryption is especially problematic for low-entropy data. High-entropy data (e.g. photos or videos) will likely always be unique and linkable and is thus less compatible with anonymous data sharing with n-PEP-like systems by nature. Low-entropy data (e.g. year of birth, nationality, gender, test results), however, is not linkable. Using box encryption for indistinguishable low-entropy data then introduces linkability that we would not have otherwise.

As a solution, when sharing *larger data* (for example, data that exceeds the available 16-byte message space, see section A), one could consider this data to be of such high entropy that it will always be linkable by nature, and thus safely use box encryption for efficiency and accept the linkability that is already present. However, for smaller low-entropy data that *does* fit one message, no box encryption should be introduced.

More generally, when building upon n-PEP, one should be cautious to avoid linkability in attributes whenever possible. Attributes should be split up into individual atomic pieces as much as possible, and subsets of data should be kept as small as possible, to prevent them from unintentionally containing (quasi-)identifiers. For example, survey responses should preferably be stored as individual attributes for every individual answer to a question, instead of (for example) a PDF export of the whole survey. This way, every answer can be individually shared.