Towards A Toolkit for Utility and Privacy-Preserving Transformation of Semi-structured Data Using Data Pseudonymization

Saffija Kasem-Madani, Martin Wehner, and Michael Meier

University of Bonn, 53113 Bonn, Germany,
{kasem, wehner, mm}@cs.uni-bonn.de,
https://net.cs.uni-bonn.de/wg/itsec/staff/

The final publication is available at https://link.springer.com/.

Abstract. We present a flexibly configurable toolkit for automatically producing pseudonymizations of data that keep certain utility. We follow a confidentiality-by-default principle. I.e., we identify utility requirements for the plaintext data and provide appropriate utility options in the pseudonymization. All remaining data is kept confidential. This stays in contrast to common pseudonymization techniques that replace only personal or sensitive data of a dataset with pseudonyms, while keeping any other information in plaintext.

Keywords: Pseudonymization, privacy, anonymity, data analysis, data utility

1 Introduction

In times of mass surveillance and mass data collection and storage, preserving the privacy of individuals is of interest. For that, privacy enhancing techniques have been studied. On stored data, the goal of applying privacy-enhancing techniques is to keep the contained privacy-relevant information confidential. On the other hand, the data should keep some of its original properties for fulfilling the use case. Consider the following application scenario: multiple sensors want to share log file content with a centralized analysis entity for obtaining a broader view on the security situation in their field. They want to hide the privacy-relevant information contained in the log file data. On the other hand, they are interested in the results of the analysis the centralized analysis entity would provide. Hence, they want to keep some certain utility of the data. In order to solve the resulting conflict between privacy and utility requirements, the sensors agree with the analysis entity on the required utility options the data should fulfill. Then, they transform the data into a data pseudonymization with utility options that meet the formulated requirements.

Another motivating example is the data processing of IoT devices in smart buildings. Consider a simple smart building that consists of actors holding one of the roles *system administrator* and *employee*. The sensors include an *access barrier* to the building that permits entrance using an employee's smart card. For that, a *smart card reader* is included. When an employee makes use of the smartcard reader, it gathers *working time data* with the employee's ID eID, the time of entrance t_e and the time of leave t_l, storing (eID, t_e) and (eID, t_l), respectively. For simplicity, we assume that presence time equals working time. On receiving a signal from the smartcard reader's site, it immediately activates the access barrier to allow for entrance or leave. The data is stored for three purposes: Working time data is collected for tracking the total hours of work of each employee (purpose 1) and for reproducing the exact working times for conflict resolution (purpose 2). The times employees enter and leave the building are also collected to be able to reconstruct the presence of individuals in certain time intervals, e.g. in case of theft detection (purpose 3). While some of the data is processed and stored locally on sensors side's registers, e.g. the signals the barrier receives for activation, some other data, e.g. the working time information is sent to a centralized database. Despite the fact that not all the data is collected and stored on a central system, the administrators are allowed to access the data. Obviously, the collected data is person-identifiable and prone to be misused for other purposes. A curious administrator may use the working time data to infer the daily routine of an employee, including habits like starting to work the same time every day [1]. This clearly contradicts the stated purposes the data have been collected for. Moreover, the use of data containing person-identifiable information is legally restricted [9].

In this work, we present a tool that transforms data into a representation that meets certain, purpose-specific utility requirements without revealing the privacy-relevant plaintext. We call that data transformation *pseudonymization with utility options*. Our contribution is as follows:

1. We define a categorization of utility requirements a pseudonymization can meet.
2. We present an XML-based policy language that allows for a precise definition and machine-readable formulation of the defined utility requirements in a so-called utility policy.
3. We present a pseudonymization toolkit that allows for
   - a user-friendly definition of utility requirements using the XML policy language;
   - a transformation of a given file that contains semi-structured data into a data representation in an XML structure that can be referenced by a utility policy;
   - generating a pseudonymization with utility options according to the utility policy.

Note that the goal of this work is not to trade-off privacy for utility. We aim at providing a pseudonymization that makes it difficult to an attacker to re-

trieve privacy-relevant information. At the same time, the technique we present in this work produces pseudonymizations that are utility-preserving. This is done by carefully selecting utility options that a pseudonymization of a dataset under consideration should provide. Compared to providing access to the plaintext data, the resulting pseudonymization keeps most of the contained information confidential. Compared to sanitizing data by identifying and removing any privacy-relevant information, our technique provides only information that is necessary for the computation. This makes it harder for an attacker to use such a pseudonymization to re-identify persons.
The rest of this work is structured as follows: After introducing notions and cryptographic facts required for this work in Section 2, related work is reviewed in Section 3. We describe the construction of pseudonyms with utility options in Section 4. The architecture of the pseudonymization toolkit is described in Section 5. In Section 6, the security requirements for a system that processes and stores pseudonymizations is discussed. Finally, the work is concluded and future work is discussed in Section 7.

2 Preliminaries

To enhance the understanding of the approach presented in this work, we shortly introduce the relevant basic notions.
We consider a dataset D being a semi-structured set of plaintext data entries d_i, $1 \leq i \leq n$, where n is the number of data entries in D. Each data entry consists of data items d_{ij}, $j \in \{1, \cdots, m_i\}$, where m_i is the number of data items of the data entry d_i. A pseudonym of a data item d_{ij} contained in a data entry $d_i \in D$ is a sequence of l_{ij} utility tags, i.e. $(u_1(d_{ij}), \cdots u_{l_{ij}}(d_{ij}))$. A utility tag is a sequence of possibly multiple strings. Its construction depends on the utility it is intentioned to represent. The set of all pseudonyms of all data items d_{ij} and all data entries d_i contained in D is the pseudonymization superset of D,

$$\mathcal{P}(D) = \bigcup_{i=1}^{n} \bigcup_{j=1}^{m_i} p(d_{ij}).$$

The set of pseudonyms that fulfills a subset of utility requirements is called a pseudonymization $P(D)$. $P(D) \subseteq \mathcal{P}(D)$. Depending on the utility options required, cryptosystems may be used to generate pseudonyms. A symmetric cryptosystem is a cryptosystem that utilizes one key k for encryption and decryption. In an asymmetric cryptosystem, a public key k_{pub} and a corresponding, mathematically connected private key k_{priv} are used for encryption and decryption, respectively. We call original data plaintexts, and encrypted data ciphertexts. A cryptosystem is called deterministic if, given a plaintext p and a key k, the output is always the same ciphertext $c(p)$, independently from the execution of the cryptographic algorithm. Its output only relies on the given input plaintext and the key. Otherwise, the cryptosystem is called probabilistic [12]. A homomorphic cryptosystem produces outputs that allow for executing operations on

them. These operations produce encrypted results of corresponding homomorphic operations on the underlying plaintexts. If the homomorphic operation on the plaintexts is an addition or multiplication, the cryptosystem is called additively homomorphic or multiplicatively homomorphic, respectively. Homomorphic cryptosystems that allow for the execution of simple operations, like addition or multiplication, are called partially homomorphic (PH). Somewhat and threshold homomorphic cryptosystems (SWH) allow for the execution of functions of limited deep on the ciphertexts [4] [3]. Fully homomorphic cryptosystems (FH) produce ciphertexts that can be used for arbitrary computation [11]. Due to their impracticability for our use cases, we are not considering SWH FH cryptosystems here and omit an explanation.
The AES is an example of a symmetric, deterministic cryptosystem [7]. The Paillier cryptosystem is an asymmetric, probabilistic, additively homomorphic cryptosystem [18]. The RSA cryptosystem in its unpadded version is asymmetric and deterministic [20]. The ElGamal cryptosystem is asymmetric and probabilistic [10]. Both RSA and ElGamal cyptosystems are multiplicative homomorphic.

3 Related Work

Saving some specific utility of structured data while keeping the plaintext content confidential has been well-studied for different usage scenarios. For SQL databases, Popa et al. have introduced CryptDB [19], a confidentiality-preserving database system that enables for SQL querying encrypted database content. For that, it provides certain, database-utility preserving encryption schemes. In contrast to our work, it only encrypts database entries that have been identified as sensitive. Any other data is kept in plaintext. Among other reasons, this increases the probability of successful inference and correlation attacks [17].
LidSec [13] is a use-case independent pseudonymization framework for the preparation of textual data for data sharing. The data consists of so-called entities of possibly multiple features. One can choose for each entity feature whether it should be kept or removed. Other possibilities include suppressing, i.e. replacing the value of a feature with a pseudonym, or removing an entity completely. Different data formats can be used. In contrast to our work, the pseudonyms generated here provide a very limited variety of utility options, e.g. the ability to check whether underlying plaintexts are equal. For more utility, the affiliated data has to be represented in plaintext.
FLAIM [23] is an open-source log data sanitization tool for privacy-respecting information sharing. Sanitization rules are formulated in an XML anonymization policy language and are considered as trade-offs between log data utility and the confidentiality of the privacy-related information contained in the log data. Use cases include the generation of shared test data as well as security and network analysis purposes. For those purposes, the tool modularly supports multiple log file formats. FLAIM provides a set of sanitization routines, called anonymization algorithms. An algorithm is applied to a value addressed in a policy rule. Depending on the defined sanitization, the value is replaced by a

truncated value, a permutation, a hashed value, an HMAC, a value that results from sanitizing a substring, or a value that represents the order of the plaintext value w.r.t. an ordering relation. The remaining utility that is expected to be provided by a sanitized log file cannot be described in a sanitization rule.
In contrast to our work, the FLAIM rules define utility at a highly technical, very use-case dependent level. They do not immediately provide information about the knowledge a curious attacker can gain by accessing the sanitized data. Moreover, only selected components of the data are sanitized. Instead of keeping the remaining data confidential, it is provided in plaintext. This makes it hard to estimate the privacy risk of sharing log files sanitized using FLAIM. All rules define replacing data items that preserve properties that make the properties of the replacing data item immediately accessible by accessing the replacing string. There is no possibility of restricting that access to certain purposes. This implies that access to a sanitized log file comes with a possibly large amount of information about the plaintext data without further control.
Autocrypt [24] is a tool that utilizes partially homomorphic encryption (PHE) to enable an untrusted web server to execute a restricted number of trusted operations on sensitive content. For this, it automatically transforms standard UNIX utilities into representations that can be used for homomorphic operation on its variables and values. In the described use case of an untrusted web server virtual machine and a trusted hypervisor, it includes the possibility to decrypt values and re-encrypt them using a different PH cryptosystem to achieve certain utility. Autocrypt can be seen as a compiler that enables programs to process pseudonymizations with utility options that match the requirements of the program.
Several privacy policy languages for various different purposes have been introduced [16][14][25]. To the best of our knowledge, none of the introduced languages can be utilized for the definition of fine-grained utility requirements for pseudonymization.

4 Utility Requirements and Construction of Pseudonyms with Utility Options

Given a semi-structured data file and the utility requirements as an input, the pseudonymization toolkit automatically generates and outputs a pseudonymized data file, called pseudonymization.
To construct a pseudonymization of a semi-structured set of data entries D, the purposes of processing must be identified. Based on the analysis of the purposes, the utility requirements for each d_{ij} are defined. For each d_{ij}, one or more utility tags will be constructed based on the defined utility requirements. This is done by applying appropriate mechanisms to d_{ij}. The resulting pseudonym $p(d_{ij})$ is then a sequence of all constructed utility tags. For each utility requirement, $p(d_{ij})$ includes a utility tag $u_l(d_{ij})$ that offers the corresponding utility option of d_{ij}. In case cryptographic parameters are required, they will be included in the belonging utility tags.

4.1 Utility requirements

In the following, we present a classification of the utility options a pseudonym can support to meet corresponding utility requirements.

Linkability A pseudonym $p(d_{ij}) \in P(D)$ of a plaintext $d_{ij} \in D$ is linkable with respect to a relation r, i.e. fulfills the utility requirement "linkability w.r.t. r", if, given another pseudonym $p(d_{xy})$ that is linkable w.r.t. the same relation r, one can determine whether the underlying plaintexts d_{ij} and d_{xy} are in a certain relation r or not. This can be modeled by a function $f : (P, P) \rightarrow \{0, 1\}$ with

$$f(p(d_{ij}), p(d_{xy})) \begin{cases} 1, & \text{if } (d_{ij}, d_{xy}) \in r \quad (1) \\ 0, & \text{else} \quad (2) \end{cases}$$

Linkability w.r.t. a relation r is available on a set of pseudonyms P, if the aforementioned function f is defined on all pseudonyms of P.

Disclosability A pseudonym $p(d_{ij}) \in P$ of a plaintext $d_{ij} \in D$ is disclosable, i.e. fulfills the utility requirement "disclosability", if a mapping $p^{-1}(p(d_{ij})) = d_{ij}$ is defined for $p(d_{ij})$.

Mathematical Operations A mathematical operation $+$ is available on a set of pseudonyms P, if there is a corresponding operation $*$ that can be applied to each pair of pseudonyms $p(d_{xy}), p(d_{ij}) \in P$ with

$$p(d_{ij}) * p(d_{xy}) = p(d_{ij} + d_{xy})$$

for all plaintexts $d_{xy}, d_{ij} \in D$ with pseudonyms in P, and there is a mapping $p^{-1} : P \rightarrow D$ with

$$p^{-1}(p(d_{ij} + d_{xy})) = d_{ij} + d_{xy}$$

for all $p(d_{ij}), p(d_{xy}) \in P$.

Binding the Accessibility of the Utility Options In order to enable a system to apply a fine-grained process-based access control on the utility of a pseudonymization of a dataset, the accessibility of a utility option can be bound to certain roles a subject in the system may hold, or to a purpose that has to be fulfilled. Hence, the utility of a pseudonym $p(d_{ij})$ can only be accessed in certain cases.

In the Smart Building example stated in the Introduction, the utility requirements of the stated purposes are as follows:

- Purpose 1: "How many hours has employee x worked?"
  Utility requirements 1:

  - The accountant must be able to disclose the employee's ID eID to create the payroll.
  - The accountant must be able to calculate the working hours of each day by calculating the differences between the corresponding entrance and leaving times t_e, t_l to create the payroll.
- Purpose 2: "At what time has employee x entered and leaved the building, respectively?" (conflict resolving).
  Utility requirements 2:
  - The conflict resolver of the company must be able to disclose eID, t_e and t_l to prove the entrance and leaving times used for calculating the working hours in case an employee has not consented the payroll.
- Purpose 3: "Who of the employees has been present during a time interval T?" (theft detection).
  Utility requirements 3:
  - When a theft is detected to happened in a certain time interval $T = [t_i, t_j]$, the security responsible of the company must be able to disclose t_e and t_l to identify the presence of employees in that time interval, i.e. all employees with last $t_e < t_j$ and last $t_l > t_i$. He also must be able to disclose all eID that correspond to suspicious timestamps.

4.2 Pseudonym Construction

The pseudonyms are constructed as sequences of utility tags, where each utility tag is used to provide a certain, well-defined utility option. We give example mechanisms that can be used to generate utility tags of a pseudonym.

Pseudonyms with Linkability options Selecting an appropriate mechanism for generating pseudonyms that are linkable depends on the relation r that has to be considered. A simple example for the linkability option is linkability with respect to equality. Here, pseudonyms are generated such that for two given pseudonyms $p(d_{ij})$ and $p(d_{xy})$ of d_{ij} and d_{xy}, $f(p(d_{ij}), p(d_{xy})) = 1$ implies $d_{ij} = d_{xy}$, and $f(p(d_{ij}), p(d_{xy})) = 0$ implies $d_{ij} \neq d_{xy}$. Pseudonyms that are linkable w.r.t. equality can be obtained by generating a utility tag $u(d_{ij})$ of d_{ij} using a symmetric block cipher. For a fast and secure utility tag and pseudonym generation, AES may be selected. To prevent the disclosure of d_{ij}, the symmetric key has to be made inaccessible.
The notion for a pseudonym with a utility tag for linkability w.r.t. equality for d_{ij} would be $p(d_{ij}) = (\cdots, u_{=}(d_{ij}), \cdots)$. In case AES is used for the generation, $u_{=}(d_{ij}) = (AES_{k_{=}}(d_{ij}))$. Note that the key $k_{=}$ is not rolled out with the utility tag.

Pseudonyms with the Disclosability option To generate a disclosable pseudonym of d_{ij}, one may utilize a symmetric encryption scheme using a key k_{discl}. To obtain a unique disclosable pseudonym for each d_{ij}, randomization is included. One possibility is to append a nonce $n_{d_{ij}}$ of fixed, known length to

each plaintext under consideration before encrypting. The key is made accessible to an entity that is responsible for managing the disclosure of the plaintext of d_{ij}. This is to ensure that only making use of the disclosability option makes plaintext information available.
The notion for a pseudonym with a utility tag for disclosability for d_{ij} would be $p(d_{ij}) = (\cdots, u_{discl}(d_{ij}), \cdots)$. In case AES is used for the generation, $u_{discl}(d_{ij}) = (AES_{k_{discl}}(d_{ij}|n_{d_{ij}}), k_{discl})$. Note that the key k_{discl} is rolled out within the utility tag. There is the need for a secure handling of the key on the processing system.

Pseudonyms with the Mathematical Operation option In this work, the utility option for a utility requirement "mathematical operation" is implemented using homomorphic encryption. For the utility option of the mathematical operation "addition" on data items d_{ij} from a subset $D_+ \subseteq D$, one may apply the Paillier cryptosystem [18] using the same public key $k_{+_{pub}}$ for generating the corresponding utility tags of all the d_{ij}. The resulting pseudonym p_{ij} includes a utility tag $u_+(d_{ij})$ that can be homomorphically added with other utility tags $u_+(d_{xy})$. The result of the homomorphic addition is an encrypted sum. To decrypt the sum, access to the private key $k_{+_{priv}}$ that corresponds to $k_{+_{pub}}$ is required.
The notion for a pseudonym with a utility tag for the addition operation for d_{ij} would be $p(d_{ij}) = (\cdots, u_+(d_{ij}), \cdots)$. In case the Paillier cryptosystem is used for the generation, $u_+(d_{ij}) = (Paillier_{k_{+_{pub}}}(d_{ij}), k_+ = (k_{+_{pub}}, k_{+_{priv}}))$. Note that the key $k_+ = (k_{+_{pub}}, k_{+_{priv}})$ is rolled out within the utility tag. Due to the construction of homomorphic cryptosystems, $k_{+_{priv}}$ can be used to decrypt the pseudonyms and hence, allows for disclosure of the plaintexts d_{ij}. $k_{+_{pub}}$ can be used to malleableize $u_+(d_{ij})$, i.e. for a known plaintext x, generating a utility tag $u_+(x)$ and homomorphically add it to a given $u_+(d_{ij})$, yielding $u_+(d_{ij} + x)$. Subtracting x from the decrypted sum will reveal d_{ij}. Thus, there is the need for a secure handling of the public and private key on the processing system.
For the utility option of the mathematical operation "multiplication", the ElGamal cryptosystem is utilized similarly.

Pseudonyms with utility options bound to roles or purposes To bind the accessibility of a utility option of a pseudonym $P(d_{ij})$ to a role or purpose, the matching utility tag $u(d_{ij})$ is probabilistically encrypted. The decryption key is made accessible only to a subject holding that certain role, or is proving to fulfill the dedicated purpose, respectively. This additional encryption introduces a layer of access control on the pseudonyms. To obtain a probabilistic encryption of a utility tag, we use a symmetric block cipher, e.g. AES, together with a nonce of fixed known length for generating each utility tag. The goal of using probabilistic encryption is to prevent a possible attacker from detecting duplicate plaintexts by checking encrypted utility tags and hence, gaining a benefit that might rise the success probability of a correlation attack using background knowledge.

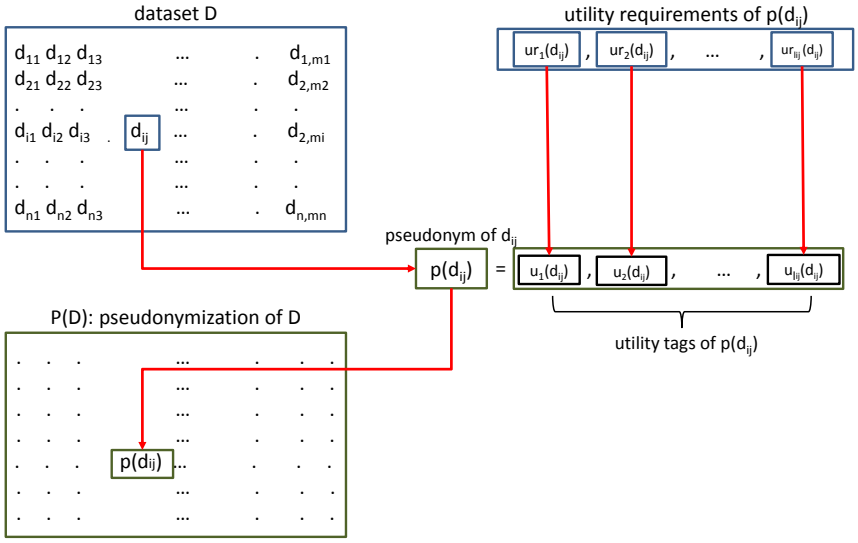

Fig. 1. Structure of a pseudonymization with utility options.

The key used for role or purpose binding of the availability of a utility option of a utility tag must be securely stored and processed in the system.

Summarizing, a pseudonymization $P(D)$ of a dataset D can be considered as a set

$$P(D) \subseteq \bigcup_{i=1}^{n} \bigcup_{j=1}^{m_i} p(d_{ij}).$$

The utility options of each d_{ij} addressed in the utility policy are represented in the pseudonymization as a sequence of utility tags. This results in a pseudonym of d_{ij} being a sequence of utility tags $u_k(d_{ij})$, i.e.

$$p(d_{ij}) = (u_1(d_{ij}), \cdots, u_{l_{ij}}(d_{ij})),$$

where l_{ij} is the number of utility tags required for meeting all utility requirements defined for $p(d_{ij})$. I.e., l_{ij} equals the number of utility requirements. The strings required to offer a single utility option are summarized in one utility tag. Fig.1 shows the structure of a pseudonymization with utility options.
Note that for each utility tag, corresponding cryptographic parameters may be required for security reasons. In order to prevent a misuse of the utility options offered by a pseudonymization $P(D)$, the parameters have to be treated carefully in the processing systems.
Note that generating pseudonyms with utility options may have side effects that lead to a disclosure of unwanted information by combining the knowledge gained out of different utility options. For that, the definition of the utility requirements has to be done very carefully and respecting the principle of data minimization and purpose binding [9] together with a strict access control.

In the example of the Introduction, data produced in a simple smart building could be pseudonymized according to each described utility requirement stated in section 4.1. This would result in utility tags constructed as follows:

- Req. 1: Tracking the total number of hours of an employee.
  Utility tags:
  - mID is probabilistically encrypted using a purpose-specific key k_1 that is only accessible for that purpose. The result is $e1_{k_1}(mID)$.
  - Every working day, t_e and t_l are used to generate pseudonyms that preserve distances within a limited time interval, e.g. using the functions described in [15]. The result is $ed(t_e)$ and $ed(t_l)$, respectively. As soon as the smartcard reader gathers an employee's leave, it calculates the difference of $ed(t_e)$ and $ed(t_l)$, yielding $ed(t_l - t_e)$. For purpose binding, the result is encrypted using k_1 to $e1_{k_1}(ed(t_l - t_e))$.
- Req. 2: Resolving conflicts about the working hours between an employee and the company:
  Utility tags: mID, t_e and t_l are probabilistically encrypted using a purpose-specific key k_2 that is only accessible for that purpose. The result is $(e1_{k_2}(mID), e1_{k_2}(t_e), e1_{k_2}(t_l))$. On conflicts, only resolving subjects in the system can access the key and decrypt and access the values.

– Req. 3: Suspect identification after theft detection:
  Utility tags: mID, t_e and t_l must be probabilistically encrypted using a purpose-specific key k_3. The result is $(e1_{k_3}(mID), e1_{k_3}(t_e), e1_{k_3}(t_l))$. To identify suspects, only security responsible subjects can access the key and decrypt and access the values.

The pseudonymization is then

– for the time stamps t_e and t_l:
  $u_1(t_e, t_l) = (e1_{k_1}(ed(t_e)), e1_{k_1}(ed(t_l)), e1_{k_1}(ed(t_l - t_e)), k_1))$
  $u_2(t_e, t_l) = (e1_{k_2}(t_e), e1_{k_2}(t_l)), k_2)$. $u_3(t_e, t_l) = (e1_{k_3}(t_e), e1_{k_3}(t_l)), k_3)$. Note that each value of the utility tag is generated as soon as the corresponding plaintext value occurs.
  The result is $p(t_e, t_l) = (u_1(t_e, t_l), u_2(t_e, t_l), u_3(t_e, t_l))$;
– for the employee's ID eID: $p(mID) = (u_1(mID), u_2(mID), u_3(mID))$.
  $u_1(mID) = (e1_{k_1}(mID), k_1)$,
  $u_2(mID) = (e1_{k_2}(mID), k_2)$, and
  $u_3(mID) = (e1_{k_3}(mID), k_3)$;

For all utility tags generated, k_1 is only accessible to the accountant for payroll generation, k_2 is only accessible to the conflict resolver in case of conflicts, and k_3 is only accessible to the security responsible in case of theft detection.

5 Architecture

The pseudonymization toolkit consists of a policy builder, an input transformation tool, and a pseudonymization tool that utilizes pseudonymization functions. Given a semi-structured data file and the utility requirements as an input, the pseudonymization toolkit automatically generates and outputs a pseudonymized data file.

The Transformation Tool

The toolkit accepts data of arbitrary semi-structured data formats as an input. In order to allow the pseudonymization tool to address the data items and apply the corresponding pseudonymization rules on each addressed data item, the transformation tool generates an XML structured representation of the input dataset. It identifies the data entries and generates an XML node for each data entry. Each data item of a data entry is identified and a corresponding sub-node is generated in the XML representation.
The transformation tool includes Python scripts [21] as plug-ins for different semi-structured data formats, including JSON [6], YAML [2], CSV [22] and the content of electronic health records written in HL7 compatible formats [8]. In the example given in Table 4.2, a CSV file consisting a column of timestamp-representing values is transformed to an XML structure. The first row of the CSV file is interpreted as containing the identifier of each data item of a row. Each row is considered to be a data entry consisting of possibly multiple data items.

CSV	XML
time 422875 278522	<dataset id="h1"> <dataentry id="0"> <time>422875</time> </dataentry> <dataentry id="1"> <time>278522</time> </dataentry> </dataset>

Table 1. Example of data represented in CSV and the corresponding XML transformation.

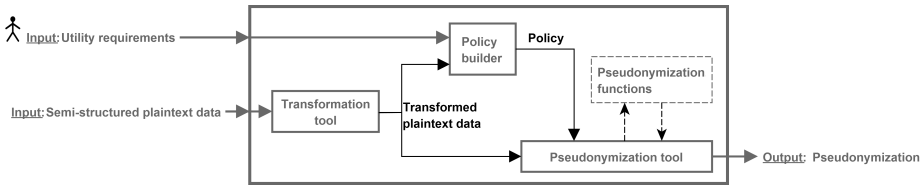

Fig. 2. Dataflow of the toolkit.

The Policy Builder

The policy builder is the human interface to the toolkit. It provides a GUI for passing the file that contains the plaintext data D and inserting the utility requirements that have to be provided by each data item. With the input, it generates a machine-readable policy using the specific XML based policy language.

The Policy Language

A utility policy has two purposes. Firstly, it serves as a machine-readable, yet human-comprehensible documentation of the utility requirements for a pseudonymization of a dataset D. Secondly, the pseudonymization tool infers the configuration required for generating a pseudonymization that meets the formulated utility requirements from the policy. Here, a configuration of the pseudonymization tool is the selection of appropriate mechanisms and parameters, e.g. cryptographic keys, to generate an appropriate pseudonymization of D.
For the example given in the Introduction, the utility requirements formulated in Section 4.2 are formalized as a policy using the policy language.

Addressing a dataset and data entries The policy language consists of XML-based syntax. A utility policy has the parent tag `<utility_policy>`. The

dataset is addressed with the child tag `<dataset id="f">`, where `f` is the identifier of the dataset's file. Each data entry d_i that contains data items to be represented in the pseudonymization by pseudonyms is addressed by an annotating child tag `<dataentry>`. The data item d_{ij} is represented by the XML tag of the column it belongs to. To define a utility requirement for a single d_{ij}, the `id` attribute of the tag `<dataentry>` is set to the number of the containing data entry. We call this referencing method individual addressation. To address all data items of the same type in a dataset, e.g. all data items of the type `<time>`, the tag `<time>` is used. The `id` attribute of `<dataentry>` is set to `all`. We call this referencing method tag-based addressation. Note that a data item can only be referenced by its position and tags, and not by its value. To address data by value-based properties, appropriate tagging is required.

```
<utility_policy id="all">
   <dataset id="h1">
        <dataentry id="all">
      <time>
         <utility>
            <option>
             mathematical operation
             </option>
            <operation>
            addition
            </operation>
            <binding type="role">
             analyzer
            </binding>
         </utility>
      </time>
     </dataentry>
   </dataset>
</policy>
```

Listing 1.1. Example of an XML utility policy for the dataset of table 4.2.

Formulation of the policy rules The rules that define the utility requirements for a pseudonym of d_{ij} are annotated with the child tag `<utility>`.

Utility options. A `<utility>` rule contains the child tag `<option>` that defines whether the utility option is of the type linkability, disclosability or mathematical operation. To annotate these utility options, the values `linkability`, `disclosability`, or `mathematical operation` are included, respectively.
For denoting the relation of linkability options, the child tag `<relation>` of `<utility>` is used. The child tag `<operation>` indicates the intended mathematical operation.

Binding a utility option to a role or purpose. The utility option defined in a `<utility>` rule can be bound to a role or purpose using the child tag `binding` of `utility`. For role binding, the `binding` is extended with an attribute `type`. The value of `type` is either set to `role` or `purpose`. The value of the `<binding>` tag is set to a system-based identifier of the intended role or purpose, respectively.

Example In listing **??**, an XML utility policy is exemplified. The scope of the example policy is a dataset with the identifier `h1`. Any data entry d_{ij} of the dataset under consideration that is tagged with `<time>` is pseudonymized in a way that the resulting pseudonym of the data entry d_{ij} includes a utility tag that allows for performing the mathematical operation "addition" on it. This utility option is bound to the role *analyzer*. Any data entry of the dataset `h1` that is tagged with a tag different from `<time>` is omitted and not included in the resulting pseudonymization. A utility policy can be addressed by extending the `<utility_policy>` tag with the `id` attribute.

The Pseudonymization Tool

The input of the pseudonymization tool is a utility policy file and an XML structured input plaintext dataset file. The tool is configured based on the utility requirements. For each utility requirement, it selects an appropriate pseudonymization function from the available pool of pseudonymization functions. It applies the functions to each data entry mentioned in the policy and writes the resulting output into an XML structured output file. Only data entries with matching rules in the utility policy are pseudonymously included in the pseudonymization output file. For the example described in Section 4.2, the pseudonymization tool selects a function that generates Paillier-encrypted ciphertexts as utility tags of the plaintext values. Note that the XML tags of the input file are not considered to be pseudonymized. The resulting data pseudonymization file content is listed in the example of Listing **??**.
Note that the provided mathematical operation "addition" is homomorphic and requires access to the corresponding Paillier public and private key. For that, the key is included in the utility tag. On deployment, it must be ensured that the key is stored and accessed securely.

```
<dataset id="h1">
   <dataentry id="0">
      <time>
      bfYUlqPrasuAdDN0grrBzuLalY
      ...
      Bo3IQw6PhGJqEdvac/oZZ0gZ
      </time>
   </dataentry>
   <dataentry id="1">
      <time>
      23L/n3AzjS1M0Gou3DaA4ak32Z
      ...
      AG9j7KnwHg5QJu4TgjX4Ut89
      </time>
   </dataentry>
</dataset>
```

Listing 1.2. Pseudonymization of the dataset of Table 4.2 according to the utility policy of Listing **??**.

6 Security Requirements

The goal of this work is to present a flexible, almost use-case independent pseudonymization toolkit. The system architecture must provide means that provide security against a realistic, well-defined attacker model. Here, we assume the honest-but-curious adversarial model. For that, the system architecture must fulfill the following conditions:

Data minimization. Depending on the usage scenario, (possibly multiple) parties may be involved in the definition of the utility requirements a pseudonymization

$P(D)$ should fulfill. We assume that the parties agree on the minimum set of utility requirements possible. This includes that the data holder would only provide pseudonyms of data entries of D with utility options that are required for fulfilling the computation purpose. I.e., the parties under consideration strictly follow the principle of data minimization.

Access control on the pseudonyms. Whenever required, the parties of the system agree on clear purposes the utility options should be used for. This implies purpose-based access control on the utility tags of a pseudonym, i.e. purpose binding. If necessary, the utility options should be made accessible to certain, well-defined roles. This lead to a role-based access control on the utility tags of the pseudonyms. The access control comprises keys, salts and nonces contained in utility tags. We assume that the system architecture provides means of secure key storage and access.

Secure decryption management. The system ensures that the private keys are only used for enabling a utility of a pseudonym according to the formulated utility policy. This includes trusted means that ensure that no plaintexts of D are revealed unless it is required to meet a well-defined defined disclosure condition. In order to fulfill this security requirement, hardware security modules may be utilized.

7 Evaluation

The goal of our evaluation was to elaborate the overhead of using the technique described in this paper in general. For that, we have considered the time and space consumption of the pseudonymizations of different utility options during generation and usage. We have examined the utility options disclosability, mathematical operation "addition", mathematical operation "multiplication", and the mathematical operation "conditional distance preservation" [15]. Each of the utility options has been considered with binding to a purpose/role, and without binding. We have run our experiments on a Microsoft Windows 10 Home 64-bit system with a Intel(R) Core(TM) i7-4500U CPU and a 1.80 GHz clock rate, a 8 GB RAM and a hard disk drive. The results are shown in table 1.

We consider pseudonyms that consist of one single utility tag. The pseudonym generation without binding takes between 0.06 and 1.7 seconds, without key generation. With additional binding, it takes between 0.12 and 2.2 seconds. The pseudonym generation is a process that is performed once before data deployment. Thus, we consider these times acceptable.
The utility processing takes between 0.009 and 0.118 seconds. Except for the decryption-performing disclosure, all the utility processing operations take less than 0.02 seconds. Taking into account that the plaintext disclosure is considered to be done in very special cases, we consider the time consumption to be practical.
The space consumptions lie between less than 16% and 23−fold of the plaintext

	Time consumption in seconds			Space consumption	
	Pseudonym generation		Utility processing	Plaintext/pseudonym in %	
	no binding	with binding		no binding	with binding
Disclosability	0.0608	0.1207	0.1184	100.00%	115.87%
Linkability w.r.t. equality	0.0904	0.1479	$f(d_{ij}, d_{xy}) = 1$: 0.0039 $f(d_{ij}, d_{xy}) = 0$: 0.0009	147.62%	249.21%
Mathematical Op.: Addition	1.7469	2.2908	0.0151	973.57%	2455.71%
Mathematical Op.: Multiplication	0.1551	0.2373	0.0099	977.86%	2455.71%
Mathematical Op.: Conditional distance preservation	0.0808	0.1477	0.0098	290.71%	627.14%

Table 2. Comparison of time and space consumption of different utility options.

size. These are constant sizes resulting from the block lengths of the executed ciphers. In this work, we have followed the BSI[1] recommendations for key and block sizes [5]. To reduce the space magnitude, making a trade-off between security and storage consumption is required.
In our opinion, the evaluation results show the practicability of our approach.

8 Conclusions and Future Work

We have presented a toolkit for pseudonymization with utility options. It provides an easy way for the formulation of utility requirements in a machine-readable utility policy of a pseudonymization. Based on the utility policy, it generates an XML structured, the utility requirements fulfilling pseudonymization.
in this work, we have presented how the availability of the utility option of a utility tag can be bound to a specific purpose or role using symmetric encryption. Utilizing asymmetric cryptography may ease the key deployment and management. However, it may imply higher storage and processing costs. Some combinations of utility options a pseudonymization can offer may be prone to correlation attacks. Building upon the presented work, one may develop technologies that enhance privacy-respecting selections of utility requirements.
When decryption is required for making a utility option of a pseudonym available, mechanisms that ensure that the decryption key is securely accessed and utilized are required as well.
More utility requirements with corresponding utility options may be identified

[1] Bundesamt für Sicherheit in der Informationstechnik: German Federal Office for Information Security

and implemented in the policy language and the pseudonymization tool, respectively.

References

1. How To Increase the Security of Smart Buildings?, author=Wendzel, Steffen, journal=Communications of the ACM, volume=59, number=5, pages=47–49, year=2016, publisher=ACM
2. Ben-Kiki, O., Evans, C., Ingerson, B.: Yaml Ain't Markup Language (yaml) Version 1.1. yaml. org, Tech. Rep (2005)
3. Boneh, D., Gentry, C., Halevi, S., Wang, F., Wu, D.J.: Private database queries using somewhat homomorphic encryption. In: International Conference on Applied Cryptography and Network Security. pp. 102–118. Springer (2013)
4. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) Fully Homomorphic Encryption Without Bootstrapping. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. pp. 309–325. ITCS '12, ACM, New York, NY, USA (2012), http://doi.acm.org/10.1145/2090236.2090262
5. BSI: Kryptographische Verfahren: Empfehlungen und Schlüssellangen. Technische Richtlinie TR-02102-1, Bundesamt fur Sicherheit in der Informationstechnik (2017)
6. Crockford, D.: The application/json media type for javascript object notation (json), 2006a. URL http://tools. ietf. org/html/rfc4627 (2006)
7. Daemen, J., Rijmen, V.: AES proposal: Rijndael (1999)
8. Dolin, R.H., Alschuler, L., Boyer, S., Beebe, C., Behlen, F.M., Biron, P.V., Shabo, A.: HL7 clinical document architecture, release 2. Journal of the American Medical Informatics Association 13(1), 30–39 (2006)
9. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). Official Journal of the European Union L119/59 (May 2016), http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=OJ:L:2016:119:TOC
10. El Gamal, T.: A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In: Proceedings of CRYPTO 84 on Advances in Cryptology. pp. 10–18. Springer-Verlag New York, Inc., New York, NY, USA (1985), http://dl.acm.org/citation.cfm?id=19478.19480
11. Gentry, C., et al.: Fully homomorphic encryption using ideal lattices. In: STOC. vol. 9, pp. 169–178 (2009)
12. Goldwasser, S., Micali, S.: Probabilistic encryption. Journal of computer and system sciences 28(2), 270–299 (1984)
13. Heurix, J., Khosravipour, S., Tjoa, A.M., Rawassizadeh, R.: LiDSec- A Lightweight Pseudonymization Approach for Privacy-Preserving Publishing of Textual Personal Information. 2012 Seventh International Conference on Availability, Reliability and Security 00, 603–608 (2011)
14. Kasem-Madani, S., Meier, M.: Security and Privacy Policy Languages: A Survey, Categorization and Gap Identification. arXiv preprint arXiv:1512.00201 (2015)
15. Kerschbaum, F.: Distance-preserving Pseudonymization for Timestamps and Spatial Data. In: Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society. pp. 68–71. WPES '07, ACM, New York, NY, USA (2007), http://doi.acm.org/10.1145/1314333.1314346

16. Kumaraguru, P., Calo, S.: A survey of privacy policy languages. In: Workshop on Usable IT Security Management (USM 07): Proceedings of the 3rd Symposium on Usable Privacy and Security, ACM (2007)
17. Naveed, M., Kamara, S., Wright, C.V.: Inference Attacks on Property-Preserving Encrypted Databases. In: Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security. pp. 644–655. CCS '15, ACM, New York, NY, USA (2015), http://doi.acm.org/10.1145/2810103.2813651
18. Paillier, P.: Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In: Stern, J. (ed.) Advances in Cryptology, EUROCRYPT 99. Lecture Notes in Computer Science, vol. 1592, pp. 223–238. Springer Berlin Heidelberg (1999), http://dx.doi.org/10.1007/3-540-48910-X_16
19. Popa, R.A., Redfield, C.M.S., Zeldovich, N., Balakrishnan, H.: CryptDB: Protecting Confidentiality with Encrypted Query Processing. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. pp. 85–100. SOSP '11, ACM, New York, NY, USA (2011), http://doi.acm.org/10.1145/2043556.2043566
20. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM 21(2), 120–126 (1978)
21. Rossum, G.: Python Reference Manual. Tech. rep., Amsterdam, The Netherlands (1995)
22. Shafranovich, Y.: Common format and MIME type for comma-separated values (csv) files (2005)
23. Slagell, A., Lakkaraju, K., Luo, K.: FLAIM: A Multi-level Anonymization Framework for Computer and Network Logs. In: LISA '06: Proceedings of the 20th conference on Large Installation System Administration. p. 6. USENIX Association, Berkeley, CA, USA
24. Tople, S., Shinde, S., Chen, Z., Saxena, P.: AUTOCRYPT: Enabling Homomorphic Computation on Servers to Protect Sensitive Web Content. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security. pp. 1297–1310. CCS '13, ACM, New York, NY, USA (2013), http://doi.acm.org/10.1145/2508859.2516666
25. Zhao, J., Binns, R., Van Kleek, M., Shadbolt, N.: Privacy Languages: Are We There Yet to Enable User Controls? In: Proceedings of the 25th International Conference Companion on World Wide Web. pp. 799–806. WWW '16 Companion, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland (2016), http://dx.doi.org/10.1145/2872518.2890590