LACE2: Better Privacy-Preserving Data Sharing for Cross Project Defect Prediction

Fayola Peters*, Tim Menzies† and Lucas Layman‡

*Lero - The Irish Software Research Centre, University of Limerick, Ireland. Email: fayolapeters@gmail.com
†Computer Science, North Carolina State University, USA. Email: tim.menzies@gmail.com
‡Fraunhofer Center for Experimental SE, College Park, USA Email: llayman@fc-md.umd.edu

Abstract—Before a community can learn general principles, it must share individual experiences. Data sharing is the fundamental step of cross project defect prediction, i.e. the process of using data from one project to predict for defects in another. Prior work on secure data sharing allowed data owners to share their data on a single-party basis for defect prediction via data minimization and obfuscation. However, the studied method did not consider that bigger data required the data owner to share more of their data.

In this paper, we extend previous work with LACE2 which reduces the amount of data shared by using multi-party data sharing. Here data owners incrementally add data to a cache passed among them and contribute “interesting” data that are not similar to the current content of the cache. Also, before data owner i passes the cache to data owner j, privacy is preserved by applying obfuscation algorithms to hide project details. The experiments of this paper show that (a) LACE2 is comparatively less expensive than the single-party approach and (b) the multi-party approach of LACE2 yields higher privacy than the prior approach without damaging predictive efficacy (indeed, in some cases, LACE2 leads to better defect predictors).

I. INTRODUCTION

When data are insufficient or non-existent for building quality defect predictors, software engineers can use data from other organizations or projects. This is called cross project defect prediction (CPDP) [1], [2]. Acquiring data from other sources is a non-trivial task when data owners are concerned about confidentiality. In practice, extracting project data from organizations is often difficult due to the business sensitivity associated with the data. For example, at a keynote address at ESEM’11, Elaine Weyuker doubted that she will ever be able to release the AT&T data she used to build defect predictors [3]. Due to similar privacy concerns, we were only able to add seven records from two years of work to our NASA-wide software cost metrics repository [4]. In a personal communication, Barry Boehm stated that he was able to publish less than 200 cost estimation records even after 30 years of COCOMO effort.

To enable sharing, we must assure confidentiality. In our view, confidentiality is the next grand challenge for CPDP in software engineering. In previous work [5], [6], we allowed data owners to generate minimized and obfuscated versions of their original data. Our MORPH algorithm [5] reflects on the boundary between an instance and its nearest instance of another class, and MORPH’s restricted mutation policy never pushes an instance across that boundary. MORPH can be usefully combined with the CLIFF data minimization algorithm [6]. CLIFF is an instance selector that returns a subset of instances that best predict for the target class. Previously we reported that this combination of CLIFF&MORPH resulted in $\frac{7}{10}$ defect data sets studied retaining high privacy scores, while remaining useful for CPDP [6]. This is a startling result since research by Grechanik et al. [7] and Brickell et al. [8] showed that standard privacy methods increase privacy while decreasing data mining efficacy.

While useful, CLIFF&MORPH only considered a single-party scenario where each data owner privatized their data individually without considering privatized data from others. This resulted in privatized data that were directly proportional in size (number of instances) to the original data. Therefore, in a case where the size of the original data is small enough, any minimization might be meaningless, but if the size of the original data is large, minimization may not be enough to matter in practice.

In this paper we mitigate this issue with LeaF for multi-party data sharing. LeaF is based on the leader follower algorithm for clustering data (explained in §III-D2). It allows a multi-party scenario where data owners can incrementally add “interesting” data to a private cache passed among them based on the content already in the private cache. This means that the size of the privatized data is no longer dependent on the size of the original data. Instead, it will depend on the (dis)similarity of the data among different data owners, i.e. the more similar the data, the less each data owner will contribute to the private cache. We implement multi-party data sharing as an extension of CLIFF&MORPH and introduce the framework called LACE which is a Large-scale Assurance of Confidentiality Environment that allow both the single-party and multi-party methods to be used by data owners.

This paper proposes and evaluates LACE2, a multi-party privacy policy based on the following scenario. Consider the problem of l parties (data owners) $P_1…P_l$, each with local data, $x_i$. They want to work together securely to create a private cache containing pooled, minimized, and obfuscated data from all parties involved. Each data owner $P_i$ determines what data to add to the private cache based on what others have added previously. The final private cache can then be published in a public data repository such as PROMISE [9].

Note, in the rest of this paper, when referring to CLIFF&MORPH which uses a single-party privacy policy, we...
experiments and results address three research questions: 

1. **RQ1**: Does LACE2 offer more privacy than LACE1? Our definition of “more privacy” is shown in §III-E.
2. **RQ2**: Does LACE2 offer more useful defect predictors than LACE1? To measure usefulness, we compare the performance of defect predictors built with local data vs. single-party privatized data (LACE1) vs. multi-party privatized data (LACE2). Results are shown in §V-B.
3. **RQ3**: Are the systems costs of LACE2 (runtime and memory) worse than LACE1? LACE2 does more work than LACE1 (specifically, it uses an instance-based nearest neighbor method to check the data should be added to the private cache). It is therefore wise to check if LACE2 runs too slowly and outputs to many instances to be practical (§V-C).

II. Background

LACE1 and LACE2 mitigate for sensitive attribute disclosure and have been tested on cross project defect prediction. The intuition behind LACE2 is based on software code reuse. According to a study done by Selby [11], in a set of programs, 32% were comprised of reused code (not including libraries). We conjecture that data will be similar over multiple projects allowing LACE2 to reduce the amount of data each data owner contributes by adding instances that are not similar to those in the private cache. The LACE2 innovation is that it supports secure multi-party computation. The goal of this novel method is to mitigate the disadvantage of LACE1 where the number of instances each data owner contributes to the private cache is directly proportional to the number of instances in the original data set. All italicized terms are defined in this section.

A. Cross Project Defect Prediction

The usefulness of LACE2 data is measured via its utility for cross project defect prediction. CPDP is useful because local data is not always available to many software companies for defect prediction [1]. According to Zimmermann et al. [1] is due to 1) the companies may be too small and 2) the product being in its first release and so there is no past data. Kitchenham et al. [12] who studied cross versus within-company cost estimation saw problems with relying on local data: (1) the time required to collect enough data on past projects from a single company may be prohibitive; (2) collecting local data may take so long that technologies used by the company would have changed and so older projects may no longer represent current practices.

With the use of better selection tools for training data, researchers have found it possible to predict defects for software projects with insufficient data by using data from other projects [1], [2], [13]–[18]. However although the field of CPDP is useful and active, its main component is data sharing which brings up privacy concerns.

B. Privacy-Preserving Data Sharing

To understand sensitive attribute disclosure, we first offer the following definitions. **Data** consists of a set of classes which we refer to as targets $\mathcal{T}=\{t_1, t_2, ..., t_{|\mathcal{T}|}\}$. Each target $t \in \mathcal{T}$ is a tuple of attribute values representing the individual target class. Each attribute falls into one or more of the following categories:

- **Direct-identifier** – the attribute explicitly identifies an individual or project (e.g. social security number or filename).
- **Quasi-identifier (QID)** – can be used to infer a target’s identity alone or in combination with other attributes.
- **Sensitive Attribute (S)** – an attribute we do not want attackers (adversaries) to associate with a target, $t$ .
- **Dependent Attribute** – used when evaluating the utility of data via classification. In this work, utility is measured via CPDP.

Privacy is threatened by unwanted disclosure of Direct-identifiers, Quasi-identifiers, and Sensitive Attributes. Privacy threats are classified as 1) identity disclosure or re-identification, 2) membership disclosure, and 3) sensitive attribute disclosure [8], [19], [20].

When protecting a personal data from privacy threats, the goal is to prevent re-identification. Re-identification occurs when an attacker with external information such as a voters list, can re-identify an individual from data that has been stripped of personally identifiable information such as a social security number. Prominent examples of this are the re-identification of William Weld from released health-care data [21] and Thelma Arnold from the AOL search data [22].

Membership disclosure is another privacy threat that focuses on protecting a person’s micro data. It can happen if an attacker is able to confirm that the target’s data is contained in a particular data set. For example, if the data set contains information only on HIV patients, then the attacker can infer that the target is HIV-positive [20].
Sensitive attribute disclosure occurs when a target is associated with information about their sensitive attributes, such as software code complexity. For example, in the case of defect data, one attribute that might want to be kept hidden are the lines of code (loc) associated with the shared data. It is well documented that loc is highly correlated to development effort [23] and development effort is something most organizations wish to keep private (since it effects how many billable hours they can charge their clients).

In this paper, we evaluate LACE1 and LACE2 against the third privacy threat, sensitive attribute disclosure. Evaluation of LACE1 and LACE2 against re-identification and membership disclosure is left to future work. Hence neither re-identification nor membership disclosure are explored further in this work.

When a data owner releases a privatized version of their data, an attacker tries to associate a specific target to a sensitive attribute value. For instance, Table I(b) shows an equal frequency binned version of Table I(a). Equal frequency binning divides the range of possible values into \( n \) bins or sub-ranges, each of which holds the same number of attribute values. If duplicate values are placed in different bins, boundaries of every pair of neighboring bins are adjusted so that duplicate values belong to one bin only [24]. The result is Table I(b).

Table I(c) is a minimized (reduced to three instances) and obfuscated (in instance #8, wmc=(6-14] changed to wmc=[3-6]) version of Table I(b). Column headers are the C-K object oriented metrics used in the data sets studied in this paper. For an explanation of those metrics, see Table II. We assume that the sensitive attribute is loc, the dependent attribute is bug and all other attributes are quasi-identifiers except for the first column which we consider to be a direct identifier.

Given Table I(b), if an attacker knows that the wmc value of their target is in the range [3-6], then the attacker will know with 100% certainty that the sensitive attribute value for loc is in the range [58-136]. With LACE we seek to reduce the attackers' certainty with data minimization and obfuscation (of quasi-identifiers) in order to disassociate quasi-identifier values from sensitive attribute values. Therefore if the attacker instead is given Table I(c) which shows a minimized and obfuscated version of Table I(b), then the attacker with the same knowledge of wmc=[3-6] will only be 50% certain about the loc range of values associated with the target.

C. Secure Multi-party Computation

LACE2 is based on the success and failures of prior work on multi-party computation. As explained by Vaidya et al. [10], the goal of perfectly secure multi-party computation is that nothing is revealed. They offer a simple example of such a computation. Suppose we want the average age of everyone attending the ICSE conference. First, we generate a large random number \( R \) and pass it to a random attendee. The attendee adds their age and passes the sum to another attendee. The attendee adds their age and passes the sum to another attendee (selected at random). This repeats till all attendees have been sampled at which point the sum returns to the origin. After subtracting \( R \), we have the sum of the ages from which we can find the mean.

The benefits of this protocol are that, if \( R \) is kept private, then no single participant can “decode” the passed value to find the sum of the ages. Also, if the ordering of the sampled attendees is also kept private and randomized, then no pair of attendees \( a, c \) can compare their numbers to determine the age of the attendee \( b \) who was sampled between \( a \) and \( c \).

Vaidya et al. discussed an experiment with a distributed data miner (based on C4.5) that used a variant of the above multi-party computation whenever it searched data from different organizations. They declared that experiment a failure for two reasons. First, the network overhead of that approach was prohibitive. Second, this approach conducted so many queries across different sites that it was possible for pairs of sites to collude to “decode” the passed values.

Our analysis of the Vaidya et al. experiment suggests that multiple micro-queries of a distributed data source lead to poor privacy and performance. However, a single-pass random sampling approach mitigates against collusion and reduces the network traffic associated with the query. LACE2 is such a single-pass randomized query whose outcome is a private cache containing exemplars from each site.

III. LACE DESIGN AND OPERATION

A. Assumptions for LACE2

When implementing LACE2, we make the following assumptions. (i) Since data is pooled into a private cache for defect prediction, each data owner must provide data with the same features or attributes. (ii) Data involved in LACE2 are not extreme. For example, consider a case where Microsoft Windows and several small “startups” contribute to a private cache. Even with the random perturbation in MORPH (described in §III-D3), it will be obvious which defect data came

<table>
<thead>
<tr>
<th>Table I</th>
<th>An example of preserving privacy of defect data via minimization and obfuscation.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(#)</td>
<td>wmc</td>
</tr>
<tr>
<td>#1</td>
<td>11</td>
</tr>
<tr>
<td>#2</td>
<td>14</td>
</tr>
<tr>
<td>#3</td>
<td>3</td>
</tr>
<tr>
<td>#4</td>
<td>12</td>
</tr>
<tr>
<td>#5</td>
<td>6</td>
</tr>
<tr>
<td>#6</td>
<td>1</td>
</tr>
<tr>
<td>#7</td>
<td>4</td>
</tr>
<tr>
<td>#8</td>
<td>14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(#)</th>
<th>wmc</th>
<th>dir</th>
<th>loc</th>
<th>bug</th>
<th>ceb</th>
<th>rfc</th>
<th>loc</th>
<th>bug</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>6-14</td>
<td>1-4</td>
<td>0-5</td>
<td>8-24</td>
<td>21-63</td>
<td>8-63</td>
<td>2-20</td>
<td>4-20</td>
</tr>
<tr>
<td>#2</td>
<td>6-14</td>
<td>1-4</td>
<td>0-5</td>
<td>1-8</td>
<td>21-63</td>
<td>8-63</td>
<td>2-20</td>
<td>4-20</td>
</tr>
<tr>
<td>#3</td>
<td>6-14</td>
<td>1-4</td>
<td>0-5</td>
<td>1-8</td>
<td>9-21</td>
<td>0-8</td>
<td>0-14</td>
<td>1-4</td>
</tr>
<tr>
<td>#4</td>
<td>6-14</td>
<td>1-4</td>
<td>0-5</td>
<td>8-24</td>
<td>21-63</td>
<td>8-63</td>
<td>0-20</td>
<td>4-20</td>
</tr>
<tr>
<td>#5</td>
<td>3-6</td>
<td>1-4</td>
<td>0-5</td>
<td>1-8</td>
<td>9-21</td>
<td>0-8</td>
<td>0-14</td>
<td>1-4</td>
</tr>
<tr>
<td>#6</td>
<td>3-6</td>
<td>1-4</td>
<td>0-5</td>
<td>8-24</td>
<td>9-21</td>
<td>0-8</td>
<td>0-20</td>
<td>4-20</td>
</tr>
<tr>
<td>#7</td>
<td>3-6</td>
<td>1-4</td>
<td>0-5</td>
<td>1-8</td>
<td>9-21</td>
<td>0-8</td>
<td>0-14</td>
<td>1-4</td>
</tr>
<tr>
<td>#8</td>
<td>3-6</td>
<td>1-4</td>
<td>0-5</td>
<td>8-24</td>
<td>21-63</td>
<td>8-63</td>
<td>0-20</td>
<td>4-20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(#)</th>
<th>wmc</th>
<th>dir</th>
<th>loc</th>
<th>bug</th>
<th>ceb</th>
<th>rfc</th>
<th>loc</th>
<th>bug</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>6-14</td>
<td>1-4</td>
<td>0-5</td>
<td>8-24</td>
<td>21-63</td>
<td>8-63</td>
<td>2-20</td>
<td>4-20</td>
</tr>
<tr>
<td>#2</td>
<td>6-14</td>
<td>1-4</td>
<td>0-5</td>
<td>1-8</td>
<td>21-63</td>
<td>8-63</td>
<td>2-20</td>
<td>4-20</td>
</tr>
<tr>
<td>#3</td>
<td>6-14</td>
<td>1-4</td>
<td>0-5</td>
<td>1-8</td>
<td>9-21</td>
<td>0-8</td>
<td>0-14</td>
<td>1-4</td>
</tr>
<tr>
<td>#4</td>
<td>6-14</td>
<td>1-4</td>
<td>0-5</td>
<td>8-24</td>
<td>21-63</td>
<td>8-63</td>
<td>0-20</td>
<td>4-20</td>
</tr>
</tbody>
</table>

The above table shows that LACE1 and LACE2 are not extreme. For example, consider a case where Microsoft Windows and several small “startups” contribute to a private cache. Even with the random perturbation in MORPH (described in §III-D3), it will be obvious which defect data came
B. Top-Level Loop of LACE

Fig. 1 gives an overview of how LACE is executed at each data owner’s site. Each data owner takes part in the process once. The shaded box where LeaF is applied with CLIFFed data and the current private cache, highlights the difference between LACE1 and LACE2. LACE1 excludes the use of LeaF (§III-D2) and only adds CLIFFed&MORPHed data to the private cache. LACE2 uses LeaF so that data owners can use the current content in the private cache to determine what data to add to the private cache. The high-level steps involving multiple data owners are explained as follows with the process of Fig. 1 reflected from Steps 2-6:

1) The initiator (data owner) is chosen at random.
2) Data owner applies CLIFF to identify the subset of data that best represents the target classes. Only the data selected by CLIFF are used in LACE.
3) If the data owners decided to use LACE1 then go to the next step, otherwise, with LACE2, LeaF is applied to further prune the CLIFFed data and facilitate collaboration among data owners.
4) The results are then obfuscated with MORPH (§III-D3), and privacy is measured (§III-E).
5) Steps 2-4 are repeated until a user defined privacy criterion is met. Once this is achieved, the MORPHed data is added to the private cache.
6) The private cache is sent to the next randomly chosen site (data owner), where they execute Step 2. As in Step 3, if LACE1 is used then move on to Step 4, otherwise, with LACE2, LeaF is applied to further prune the CLIFFed data and facilitate collaboration among data owners.
7) The private cache moves on to the next random data owner and Steps 2-6 are repeated.
8) The protocol is complete when all data owners involved have had a chance to contribute to the private cache. This final private cache can be added to a public data repository.

The rest of this section offers further details on LACE guided by the main components in Fig. 1.

C. Inputs for LACE

LACE accepts three inputs provided by the data owner: data, privacy criteria based on the privacy threat of sensitive attribute disclosure, and the private cache which can be empty or contain privatized data from other data owners.

D. Privacy Algorithms in LACE

LACE uses the CLIFF algorithm to remove uninformative instances and the MORPH algorithm to obfuscate the remaining instances. On top of that, the LACE2 innovation is to apply the LeaF “leader-follower” algorithm to add more intelligence to what instances are selected for the private cache.

1) CLIFF for Data Reduction: CLIFF [6] assumes that tables of training data can be divided into classes. For example, for a table of defect data containing code metrics, different rows might be labeled accordingly (defective or not defective). CLIFF executes as follows:
   - For each column of data, find the power of each attribute sub-range; i.e., how frequently that sub-range appears in one class more than any other. We then find the product of the powers for each row then.
   - Remove the less powerful rows of each class, keeping 20% of the most powerful rows. We use 20% based on the results from previous work where selecting 20% of the top ranked rows provided a relatively better balance between privacy and utility. The result is a reduced data set with fewer rows.

Finding the power of each attribute sub-range is based on the BORE (best or rest) [25] algorithm. To apply BORE, first we assume that the target class is divided into one class as first and the other classes as rest. This makes it easy to find the attribute values that have a high probability of belonging to the current first class using Bayes’ theorem. The theorem uses evidence E and a prior probability P(H) for hypothesis H ∈ {first, rest}, to calculate a likelihood (hereafter, like) of the evidence selecting for one class:

\[ \text{like}(H|E) = P(E|H) \times P(H). \]

This calculation is then normalized to create probabilities:

\[ P(\text{first}|E) = \frac{\text{like}(\text{first}|E)}{\text{like}(\text{first}|E) + \text{like}(\text{rest}|E)} \] (1)

Jalali et al. [25] found that Equation 1 was a poor ranking heuristic for low frequency evidence. To alleviate this problem the support measure was introduced. Note that like(first|E) is also a measure of support since it is maximal when a value occurs all the time in every example of one class. Hence, adding the support term is just (Equation 2):

\[ P(\text{first}|E) \times \text{support}(\text{first}|E) = \frac{\text{like}(\text{first}|E)^2}{\text{like}(\text{first}|E) + \text{like}(\text{rest}|E)} \] (2)

2) LeaF for Data Selection: LeaF is based on the leader-follower algorithm [26]. It is an online, incremental technique for clustering data. The cluster centers are the “leaders” and all other instances are the “followers”. For this work we are
only interested in the leaders. The basic algorithm works as follows: First initialize cluster centers, then for each instance in the data, find its nearest center. If the distance to the center is less than a user defined distance, then update the cluster. Otherwise, create a new cluster with the instance as the center.

To define distance, we use the standard Euclidean measure recommended for instance-based reasoning by Aha et al. [27]:

$$\text{dist}(x, y) = \sqrt{\sum (x_i - y_i)^2}$$  \hspace{1cm} (3)

where $x_i$ and $y_i$ are normalized values between 0 and 1.

Leaf is applied to each instance selected by CLIFF from the data owner’s data set to determine if it should be included in the private cache. For our work, we adapt Leaf as follows. First, the cluster centers are never updated to create centroids (this saves some time in the algorithm). Second, instead of a user defined distance, we randomly select 100 instances from the initiator and find the distances from their nearest neighbor with a different class label. We use the median of these distances $d$, to determine if data from a data owner should be included in the private cache (new data is added to the cache if it falls outside of $d$). Third, prior to a new instance being added to the cache, it is first MORPHed using the method described in the next section.

3) MORPH for Data Oblfuscation: MORPH’s role in the LACE process is to obfuscate the output from either CLIFF (if LACE1 is used) or Leaf (if LACE2 is used) prior to the output’s addition to the private cache. MORPH is an instance mutator used as a privacy algorithm [5], [6]. It changes the numeric attribute values of each row by replacing these original values with MORPHed values.

MORPHed instances are created by applying Equation 4 to each attribute value of the instance. MORPH will not change an instance such that it moves across the boundary between the original instance and instances of another class. This boundary is determined by $r$ in Equation 4. A small $r$ value means the boundary is closer to the original row, while a large $r$ value means the boundary is farther away from the original row.

$$y_i = x_i \pm (x_i - z_i) \ast r$$  \hspace{1cm} (4)

Let $x \in \text{data}$ be the original instance to be changed, $y$ the resulting MORPHed instance and $z \in \text{data}$ the nearest unlike neighbor of $x$, i.e. whose class label is different from $x$’s class label. Distance is calculated using the Euclidean distance. Previously, in our work on CLIFF&MORPH [6] the random number $r$ was calculated with the property:

$$\alpha \leq r \leq \beta$$

where $\alpha = 0.15$ and $\beta = 0.35$. We use this range of values based on results of previous work [6] which produced privatized data candidates with high privacy and accurate defect prediction.

E. Measuring Privacy

To measure privacy, we use the Increased Privacy Ratio (IPR) used in our previous work [6]. Informally, it can be defined as follows. Suppose the same query is posed to a database, before and after some algorithm has tried to privatize that data. The privacy ratio is the percent of data found before that was also found afterwards:

- If that ratio is 100% then this would be an example of a very poor privacy algorithm.
- If, on the other hand, none of the data found before was found in after, then this would be an example of a very good privacy algorithm.

We report the IPR as the percent of data not found, therefore a poor privacy algorithm will have IPRs closer to 0% while a good privacy algorithm will have IPRs closer to 100%.

It should be noted that in CPDP, if the goal of the attacker is to associate a target to the number of defects then no privacy algorithm can defend against this except to generalize the values of defects for each target as done in this work. Here any number of defects are replaced with the value one.

To formally define IPR, we assume that attackers have access to privatized data (in this case, exemplars prior to joining the private cache), denoted as $(T')$ of an original data set $(T)$, and some background knowledge of non-sensitive quasi-identifier values for a specific target in $T$. We refer to the background knowledge as a query. To generate queries we use a query generator to generate queries based on what the attacker may know about a target in the original data set.

To maintain some “realism” to the attacks, a selected sensitive attribute(s) and the class attribute are not used as part of query generation; the attacker is trying to discover this information but does not know it beforehand. Here we are assuming that the only information an attacker could have is information about the non-sensitive QIDs in the data set.

To illustrate a query generator, we use an example defect data set shown in Table I(a) and Table I(b). Next, to create a query, we proceed as follows. Our inputs are a set of attributes and a query size measured as the number of attribute sub-range pairs. For this study, we use a query size of 1 since previous work [6] showed that even with an increase in query size, IPRs were comparable. From those inputs, we randomly choose an instance from the data. For this example we use row 1 in Table I(b), then randomly select an attribute from A, e.g. $wmc = (6-14)$. In the end the query we generate is, $wmc = (6-14)$. We continue this process until we have used all instances. In previous work [6] we also used 1000 unique queries as a stopping criterion because of query sizes 2 and 4. We stopped at 1000 because it would not be practical to generate and test every possible query of size 2 and 4. However, with a query size of 1 this stopping criterion is unnecessary because with equal frequency binning set at 10 bins, each attribute in the data sets used in this work, will have at most 10 sub-ranges and with the number of quasi-identifiers at 19, the most number of queries generated are 190.

Each query must also satisfy the following sanity checks:

- They must not be the same as another query.
- They must return at least one instance from the original data set.
• They must not include attribute value pairs from either the designated sensitive attribute or the class attribute.

When all the queries are generated the next steps are as follows: For each query, \( q \in Q = \{q_1, \ldots, q_Q\} \), \( G_i^* \) is a group of rows from any data set which matches \( q_i \). \( G_i \) is the group from the original data set and \( G_i^* \) is the group from the private data candidate which matches \( q_i \). Next, for every sensitive attribute sub-range in the set \( (s) = \{s_1, \ldots, s_{|S|}\} \), the most common sensitive attribute value is \( s_{\text{max}}(G_i^*) \).

Now, we define a breach of privacy as follows:

\[
\text{Breach}(S, G_i^*) = \begin{cases} 
1, & \text{if } s_{\text{max}}(G_i) = s_{\text{max}}(G_i^*), \\
0, & \text{otherwise}.
\end{cases}
\]

Therefore, the privacy level of the exemplars is:

\[
P_1 = 100 \times \text{IPR}(T^*) = 1 - \frac{1}{|Q|} \sum_{i=0}^{Q} \text{Breach}(S, G_i^*). \tag{5}
\]

IPR\((T^*)\) has some similarity to \( A_{\text{acc}} \) of Brickell and Shamitkov [8], where IPR\((T^*)\) measures the adversary’s ability to cause breaches after observing the exemplars \( T' \) compared to a baseline of the original data set \( T \). To be more precise, IPR\((T^*)\) measures the percent of total queries that did not cause a Breach.

\section*{F. Upper and Lower Bounds on IPR}

When used in conjunction with instance selection algorithms like LeaF and CLIFF, Equation 5 is a lower bound on privacy. Recall that:

• From with \( N \) projects, CLIFF and LeaF discards \( X \) rows;
• Equation 5 is applied to the remaining \( N - X \) projects.

Since data from the \( X \) discarded projects is never shared, it is fully private. Therefore, an upper bound on privacy is:

\[
P_2 = \frac{X}{N} + \frac{(N - X)}{N} \times P_1 / 100 \tag{6}
\]

where \( P_1 \) comes from Equation 5. For example, CLIFF and LeaF typically discard 80% of the data and, on the remaining data, we achieve an IPR of 80%. The resulting increased privacy is hence 0.8 + 0.2 * 0.8 = 96%.

Note that \( P_2 \) is an upper bound on privacy since it is possible that the patterns in the discarded data might repeat in the cached data. That said, given a large enough community sharing their data, there would always be some doubts about which members of the community had the exact values found in particular query.

In Table V, we take care to report the lower and upper bound (\( P_1, P_2 \)) on all our privacy results.

\section*{G. Output}

Once data have been MOPRHed and meet the data owner’s criterion of high IPR, the data are added to the private cache and either sent to another data owner or made public for CPDP. When applying the data owner’s IPR criterion, we use the conservative bound of Equation 5 rather than the more optimistic Equation 6.

\section*{IV. Experimental Setup}

\subsection*{A. Experimental Design}

These experiments are designed to address the three research questions from the introduction (§1).

First, to determine if LACE2 offers more privacy than LACE1 (RQ1), we calculate the IPRs for the privatized data produced by each method (explained in §III-E) prior to being added to the private cache. In practice, the data owner may choose to lower or raise their privacy criterion. This means that no matter how many data owners are involved in LACE2, the IPR will always be adequate for the data owner. We define adequate to be the equivalent of a data owners’ privacy criterion. In our experiments we use an arbitrary privacy criterion of 65% therefore adequate \( \geq \) 65%. Results are shown in Table V.

Second, to determine if LACE2 offers better defect predictors than LACE1 (RQ2), we baseline our work with a cross-validation experiment on local data. Cross-validation is a standard evaluation approach in Machine Learning where an experiment is repeated \( n \) times on \( m \) random subsamples of data. In other words, \( n \)-times, \( m_{\text{all}} \)-m, is treated as the training set and \( m_{\text{all}} \)-training set, is the test set. We use a 10-way cross-validation where \( n \) is 1 and \( m \) is 10 and report on the median performance (Section IV-D shows how this is measured).

Last, to determine if LACE2 consumes more processing and storage resources than LACE1 (RQ3), we measure the time (seconds) it takes for each to produce a private cache and also measure the size of the cache. Results are shown in Table VIII.

\subsection*{B. Data}

The evaluation was conducted using 17 of the Jureczko static code defect data sets [28], [29]. Table II describes the attributes of these projects and Table III lists the names of the data sets. Each instance in these data sets represents a source code class and consists of two parts: 20 independent static code attributes and the dependent attribute labeled “defects” indicating the number of defects in the class. For our work, we refer to each class as an instance. Additionally, instances with no defects are labeled as 0, and instances with one or more defects are labeled as 1. Table III also indicates that the first 10 data sets are from open-source projects while the remaining seven are from proprietary projects.

\subsection*{C. Data Mining Algorithms}

To assess the performance implications of applying our privacy algorithms, we used a k-Nearest Neighbor (k-NN) algorithm. Cover and Hart [30] describes k-NN as a simple non-parametric decision procedure which classifies an unknown instance in the category of its nearest neighbor. k-NN is one of the simplest defect predictors that can be used. It can therefore be used as a baseline for more complicated methods. A k-NN algorithm generates an estimate for a test instance by finding the mean of the \( k \) nearest neighbors in the training data. To define distance in this context, we use Equation 3.
Hence, in this work, in addition to using k-NN as a classifier but found that these learners generated unacceptably high false alarm rates. Only one of relevancy or noise filtering but those results had unacceptably high false alarm rates.

D. Performance Evaluation

We assess our privacy algorithms using (1) the IPR privacy measure (described above) and (2) the g-measure that summarizes the performance measures of Table IV. TP, TN, FP and FN are true positive, true negative, false positive and false negative respectively. Probability of detection or \(pd\) is equal to how much of the target (defective instances) are found. The higher the pd, the fewer the false negative results. The probability of false alarm or \(pf\) measures how many of the instances that triggered the detector actually did not contain the target (defects) concept. Like pd, the highest value is known as specificity (not predicting instances without defects as defective). Specificity is used together with \(pd\) to form the G-mean\(_2\) measure seen in Jiang et al. [35].

Measures such as accuracy, precision, and f-measure are not shown in our experimental results since they are poor indicators of performance for data where the target class is rare (in our case, the defective instances). This is based on a study done by Menzies et al. [36] which shows that when data sets contain a low percentage of defects, precision can be unstable. If we look at the data sets in Table III, we see that defects are rare in most cases.

V. EXPERIMENTAL RESULTS

We organize our results around the three research questions in the introduction (§I).

A. RQ1: Does LACE2 offer more privacy than LACE1?

Table V displays the median lower and upper bound results of IPRs of the data submitted to the private cache by each data

\[
\text{TABLE III}
\]

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amc</td>
<td>average method complexity: average method size as measured by the number of Java binary codes</td>
</tr>
<tr>
<td>avg_cc</td>
<td>average McCabe: average McCabe’s cyclomatic complexity seen in class</td>
</tr>
<tr>
<td>ca</td>
<td>afferent coupling: the number of classes the access the members of the specified class</td>
</tr>
<tr>
<td>cam</td>
<td>cohesion amongst classes: summation of number of different types of method parameters in every method divided by a multiplication of number of different method parameter types in whole class and number of methods</td>
</tr>
<tr>
<td>ccm</td>
<td>coupling between methods: total number of new/redefined methods to which all the inherited methods are coupled</td>
</tr>
<tr>
<td>cbo</td>
<td>coupling between objects: increased when the methods of one class access services of another</td>
</tr>
<tr>
<td>ce</td>
<td>efferent couplings: the number of classes whose methods are access by the specified class</td>
</tr>
<tr>
<td>dam</td>
<td>data access metric: ratio of the number of private (protected) class attributes to the total number of class attributes</td>
</tr>
<tr>
<td>dit</td>
<td>depth of inheritance tree: the level on which the class is positioned in the inheritance tree ((dit(root) = 0))</td>
</tr>
<tr>
<td>ic</td>
<td>inheritance coupling: number of parent classes to which a given class is coupled (includes counts of methods and variables inherited)</td>
</tr>
<tr>
<td>lcom3</td>
<td>lack of cohesion in methods: number of pairs of methods that do not share a reference to an instance variable</td>
</tr>
<tr>
<td>loc</td>
<td>lines of code: number of lines of binary code</td>
</tr>
<tr>
<td>max_ce</td>
<td>maximum McCabe: maximum McCabe’s cyclomatic complexity for class</td>
</tr>
<tr>
<td>mf</td>
<td>measure of function abstraction: number of methods inherited by a class plus number of methods accessible by member methods of the class</td>
</tr>
<tr>
<td>moa</td>
<td>measure of aggregation: count of the number of data declarations (class fields) whose types are user defined classes</td>
</tr>
<tr>
<td>noc</td>
<td>number of children: measures the number of immediate descendants of the class</td>
</tr>
<tr>
<td>npm</td>
<td>number of public methods: counts all the methods in a class that are declared as public. The metric is known also as Class Interface Size (CIS)</td>
</tr>
<tr>
<td>rfc</td>
<td>response for a class: sum of the number of methods, and the number of methods invoked within a class’s method bodies</td>
</tr>
<tr>
<td>wmc</td>
<td>weighted methods per class: the number of methods in the class (assuming unity weights for all methods).</td>
</tr>
<tr>
<td>defects</td>
<td>number of defects per class, seen in post-release bug-tracking systems. Converted to the boolean false if no defects, otherwise true.</td>
</tr>
</tbody>
</table>

Note that, in our initial experiments, we used Naive Bayes [31], Neural Networks [32] and Support Vector Machines [33] but found that these learners generated unacceptably high false alarm rates (median values of 50% or higher). Hence, in this work, in addition to using k-NN as a classifier it is also used for relevancy filtering.

In relevancy filtering [2], [15], [18], only the training data nearest to the test data is used to learn predictive models. The filter applies k=10-NN to each member of LACE2’s cache to build such a “nearest neighbor” training set. However instead of using \(k=10\), we tune \(k\) using the “Best(K)” procedure used by Kocaguneli et al. [34] to determine the best for each test set. For this study, we used k=1-NN for our relevancy filtering.

The results of relevancy filtering are then passed to noise filtering to remove outliers. For this study, we used CLIFF to for noise filtering. Note that we also experimented with using only one of relevancy or noise filtering but those results had unacceptably high false alarm rates.
Recall that we set an IPR as adequate if it was $\geq 65\%$, so that if any data owner has an IPR less the adequate they can run LACE again. However if the adequate measure is not reached, we hypothesize that for these cases the data lacks diversity and so any subset of the data are similar to all the data. For these rare occurrences, data owners can choose not to add their exemplars to the private cache. In future work we will test the utility of not adding these exemplars.

**Overall, LACE2 provides more privacy than LACE1.**

Note that the data set names in Table VI are different from Table V. To mimic true cross-project learning in these experiments, the proprietary data sets of Table V are used to build the cache of shared data, and the resulting prediction model is evaluated against the open source data sets of Table VI.

Table VI comments on the benefits of sharing. Note that LACE’s intelligent selection of training data can lead to much higher pds. Overall, in $\frac{1}{5}$ data sets, the median pd seen after learning from LACE2 was relatively higher than that learning from the local data and LACE1 data. More generally, consider the five local pd results that are less than 50% (for ant-1.7, camel-1.6, ivy-2.0, jEdit-4.1, xerces-1.3). LACE2 boosts all of these results by 15% (for xerces) to 50% (ivy-2.0).

As to pfs, increasing the probability of detection usually means some more false alarms. Hence, LACE2’s pfs are higher than those using the local data or LACE1. That said, the pfs shown here for LACE2 are not abnormally large compared to prior results (median pf median here $= 36\%$; median pf in a IEEE TSE paper=28% [37]). Also, some of those large pfs are associated with substantial pd improvements. For example, ivy-2.0’s pd,pf for local and LACE2 are (30,5) and (80,45) respectively (which is a marked improvement).
While individual results differ, there is no overall loss of predictive efficacy due to LACE2. Table VII checks for significant differences between these prediction results. In the column headers, the arrows indicate the direction to interpret the results. For example, for the pf values for LACE1→local, LACE1 has significantly worse pfs than local. When we compare LACE2 with local result, we find that for pd and g-measure, there are no significant difference in the results. However, we find that LACE2 pfs are significantly worse than local. The same can be said when LACE2 is compared with LACE1 and LACE1 is compared with local.

The interesting feature of these results is that the LACE2 results are no worse than LACE1. This is surprising since in LACE1 data owners contribute approximately three times more data than those data owners who apply LACE2 (as seen in the next section). Thus we say:

**Overall, there is no loss of predictive efficacy due to the multi-party computation of LACE2.**

**C. RQ3: Are the systems costs of LACE2 (runtime and memory) worse than LACE1?**

Table VIII shows the number of instances and the percentage of instances that are added to the private cache by each proprietary project for LACE1 and LACE2. The last row shows the median runtimes in seconds that it takes to build the final private cache. The data sets in the first column are sorted from the least number of instances to the most number of instances. We recognize that if the actual values in column two are small enough, any reduction might be essentially meaningless, but if the numbers are large, a reduction might not be enough to matter in practice.

From our results we find that this is the case with LACE1 whose reduction is solely the responsibility of CLIFF (§III-D1) which selects the top 20% of the most powerful instances in a data set. While with LACE2, in addition to CLIFF, reduces the number of instances shared by using LeafF (§III-D2), and LeafF selection is based on instances being dissimilar to those in the private cache rather than a fixed percentage. Therefore in the case of LACE1 as the data sets get larger the reduction will eventually not be enough to matter. LACE2 avoids this reality when data shared by different data owners are similar, for example, in Table VIII, prop43-ver512 and prop5-ver185 contains 2265 and 3260 instances respectfully.

**D. Threats to Validity**

As with any empirical study, biases can affect the final results. Therefore, any conclusions made from this work must be considered with the following issues in mind:

1. **Sampling bias** threatens any classification experiment; i.e., what matters there may not be true here. For example, the data sets used here comes from the PROMISE repository and were supplied by one individual. Also even though we use ten open-source data sets for CPDP (Table III) and seven to run LACE (Table III), and the data covers a large scope of applications including text/xml processing systems, search engines, source code integration/build tools, and management information systems, they are all from Java systems.

2. **Learner bias:** For building the defect predictors in this study, we elected to use k-Nearest Neighbor. We chose the k-Nearest Neighbor because its results were comparable to the more complicated algorithms [38] and can act as a baseline for other algorithms. Classification is a large and active field and any single study can only use a small subset of the known classification algorithms.

3. **Evaluation bias:** This paper uses one measure of privacy, IPR. Other privacy measures used in software engineering include guessing anonymity [39], [40], and entropy [41], [42] (discussed in §IV-D). Measuring privacy with other measures is left for future work.

4. **Order bias:** With LACE2, the order that the data owners get access to the private cache affects the amount of data that they submit to the cache. To mitigate this order bias, we run the experiment 10 times randomizing the order of the data owners each time.

5. **Input bias:** For the MORPH algorithm, we randomly select input values for a set range to determine the boundary between the an instance and its nearest unlike neighbor within which we create MORPHed instances. Since different input values can result in different outputs, we mitigate this bias with 10 runs of the experiment for LACE1 and LACE2.

E. **Relation to Other Work**

LACE2 is designed based on the privacy needs of CPDP. Other researchers in SE focus on privacy in software testing and debugging [39]-[43]. This becomes an issue when it involves: 1) Collecting user information after a software system has been deployed [41], [42]; Or 2) outsourcing the software testing to third parties (e.g. see Budi et al. [43], Taneja et al. [39] and Li et al [40]). In this case, companies do not wish to release actual cases for testing. Hence, they anonymize the test cases before releasing them to testers.

Work published by Castro et al. in 2008 [41], sought to provide a solution to the problem of software vendors who need to include sensitive user information in error reports to reproduce a bug. To protect sensitive user information, the authors used symbolic execution along the path followed by a failed execution to compute path conditions. Their goal was to compute new input values unrelated to the original input. The new input values satisfied path conditions required to make the software follow the same execution path until it failed.
As a follow-up to the Castro et al. [41] paper, Clause et al. [42] presented an algorithm which anonymized input sent from users to developers for debugging. Like Castro et al. [41], the aim of Clause et al. was to supply the developer with anonymized input which causes the same failure as the original input. To accomplish this, they first use a novel “path condition relaxation” technique to relax the constraints in path conditions thereby increasing the number of solutions for computed conditions.

In contrast to the work done Castro [41] and Clause [42], Taneja et al. [39] proposed PRIEST, a privacy framework. Unlike our work, which privatizes data randomly within “nearest unlike neighbor” border constraints, the privacy algorithm in PRIEST is based on data-swapping where each value in a data set is replaced by another distinct value of the same attribute. This is done according to some probability that the original value will remain unchanged.

Work by Taneja et al. [39], followed work done by Budi et al. [43]. Similarly, their work focused on providing privatized data for testing and debugging. They were able to accomplish this with a novel privacy algorithm called $kb$-anonymity. This algorithm combined $k$-anonymity with the concept of program behavior preservation which guide the generation of new test cases based on known ones and make sure the new test cases satisfy certain properties [43]. The difference with the follow-up work by Taneja et al [39], is that while Budi et al. [43] replaces the original data with new data, in Taneja’s work [39], the data-swapping algorithm maintains the original data and offers individual privacy by swapping values.

Software test outsourcing work by Li et al. [40], follows a similar approach to our work in privacy for CPDP (LACE1 and now LACE2 with Leaf): 1) Don’t use all the data (minimize), and 2) obfuscate data that are used. Li et al. accomplish this through the process of securing centroids using a novel combination of data mining approaches, program analysis, and privacy constraints.

F. Future Work

In the study of data privacy, modeling the adversary’s background knowledge is important to determine how private a data set is. In this paper we only focused on background knowledge specific to the original data sets. Other types of background knowledge need to be considered.

The above results need to be explored on a wider range of data sets. For example, it would be interesting to check if the above results hold for more than just defect prediction.

The runtimes reported above were generated from a single core machine simulating data being passed around a community of data owners. It is possible that a much faster parallel computation could be achieved if (a) when sending a cache, it gets dispatched to $N > 1$ other data owners; and (b) when receiving $N > 1$ caches, there is some work on combining data from different caches.

Finally, LACE2 considers all attributes somewhat equal with respect to the semantic meaning of their data. Future work would consider that some attributes (not identifiers) have higher impact and should be treated differently.

VI. Conclusions

Studies have shown that early detection and fixing of defects in software projects is less expensive than finding defects later on [44]. Organizations with local data can take full advantage of this early detection benefit by doing local defect prediction. When an organization does not have enough local data to build defect predictors, they might try to access relevant data from other organizations in order to perform cross defect prediction. That access will be denied unless the privacy concerns of the data owners can be addressed.

This paper has presented LACE2, a novel private multi-party sharing protocol for CPDP. LACE2 is an extension of our prior system (LACE1) [6] and offers and additional method for data sharing with significant improvement over our LACE1. LACE2 is a multi-party computation that works incrementally on sub-samples of the data. The experiments of this paper show that this approach generates higher privacy than LACE1 without damaging predictive efficacy. Better yet, measured in terms of runtimes and how much data must be based around the network, LACE2 is not more expensive than LACE1.

We hope that this result encourages more data sharing, more cross-project experiments, and more work on building software engineering models that are general to large-scale systems.

ACKNOWLEDGMENT

This work was partially funded by a National Science Foundation CISE medium grant (#1302169), Science Foundation Ireland grant 10/CE/11855 and by the European Research Council (Advanced Grant 291652 - ASAP).