Privacy-preserving record linkage using local sensitive hash and private set intersection

Allon Adir[1], Ehud Aharoni[1], Nir Drucker[1], Eyal Kushnir[1], Ramy Masalha[1], Michael Mirkin[2⋆], and Omri Soceanu[1]

[1] IBM Research - Haifa
[2] Technion - Israel Institute of Technology

Abstract. The amount of data stored in data repositories increases every year. This makes it challenging to link records between different datasets across companies and even internally, while adhering to privacy regulations. Address or name changes, and even different spelling used for entity data, can prevent companies from using private deduplication or record-linking solutions such as private set intersection (PSI). To this end, we propose a new and efficient privacy-preserving record linkage (PPRL) protocol that combines PSI and local sensitive hash (LSH) functions, and runs in linear time. We explain the privacy guarantees that our protocol provides and demonstrate its practicality by executing the protocol over two datasets with 2^{20} records each, in $11-45$ minutes, depending on network settings.

Keywords: Privacy-Preserving Record Linkage, Entity Resolution, Private Set Intersection, Local Sensitive Hash, Information privacy, Data security and privacy, Secure two-party computations

1 Introduction

Entity resolution (ER) is the process of identifying similar entities in several datasets, where the datasets may belong to different organizations. While these organizations would like to join hands and analyzes the behavior of matching customers, they may be restricted by law from sharing sensitive client-data such as medical, criminal, or financial information. The problem of matching records in two or more datasets without revealing additional information is called privacy-preserving record linkage (PPRL) [11] or blind data linkage (BDL) [10] and is the focus of this paper. A survey of PPRL methods is available in [18]. The importance of finding efficient and accurate PPRL solutions can be observed, for example, in the establishment of a special task team by the Interdisciplinary Committee of the International Rare Diseases Research Consortium (IRDiRC) to explore different PPRL approaches [1].

The PPRL problem is a generalization of the well-studied private set intersection (PSI) problem in which two parties with different datasets would like to

---

⋆ The work for this paper was done while Michael Mirkin was with IBM Research.

know the intersection or the size of the intersection of these datasets without revealing anything else about their data to the other party. Examples for PSI solutions include [5,6,12,20,30,32]. With PSI, the two parties compute the intersection of their respective sets, which can be used to identify matches by looking for records that share the same identifying field e.g., PSI over social security numbers (SSNs). However, in reality, such identifying fields do not always exist, and even when they do exist, their content may be entered incorrectly or differently. For example, consider two parties that perform PSI on entity names. A single user may register himself in different systems under the names: 'John doe', 'John P Doe', 'john doe', just 'John', or even 'Jon ode' by mistake. A general PPRL solution may attempt to consider all of the above names as matching.

In some cases, more than one data field is used to match two records, e.g., first name, last name, addresses, and dates of birth. These fields are known as quasi-identifiers (QIDs), which may hold private information. In this paper, we assume that the parties are allowed to learn data by matching QIDs. In other cases, one can use a masking method e.g., as in [25] to maintain the users' privacy.

Non-exact matching is commonly performed using ER solutions that employ a local sensitive hash (LSH) function. Unlike cryptographic hash functions, this technique permits collisions by deliberately hashing similar inputs to a single digest. For example, consider a hash function that hashes all the above names to a single digest value or to lists of digests with non-empty intersection. Different LSH functions with different parameters allow us to fine-tune the results in different ways. We provide more details in Section 2.2.

Unfortunately, few practical protocols exist that can securely perform such "fuzzy" record linkage without revealing some private data of the parties, and do so in a linear time frame. See Section 1.1 for a review of the different approaches. Many involve a third-party (e.g., [23]), which we aim to avoid, while other works do not provide a thorough leakage analysis that would help evaluate the security of the solution. To this end, we constructed a new and efficient PPRL solution that runs in $\mathcal{O}(n)$. We describe its performance and discuss its security characteristics.

The goal of our solution is to compose a PSI with an LSH function. The dataset fields are first locally hashed by both parties using the LSH and then checked for matches using PSI. The choice of PSI algorithm can only affect the performance (latency and bandwidth) of our solution but does not affect the amount of leaked information that can be tuned using the different parameters of the LSH. Figure 1 illustrates a high-level view of our solution.

Our contribution. Our contributions can be summarized as follows:

- We introduce a novel and efficient PPRL protocol that combines LSH and PSI, and analyze its security against semi-honest adversaries. It does not involve third parties. Specifically, due to the use of LSH, our protocol has a low probability of revealing the data of non-matched records and thereby provides better privacy guarantees.

– We implemented the model and suggest several lower-level and higher-level optimizations.
– We evaluated our implementation over a dataset with 2^{20} records and demonstrated its practical advantage when the execution took $11 - 45$ minutes, depending on network settings.
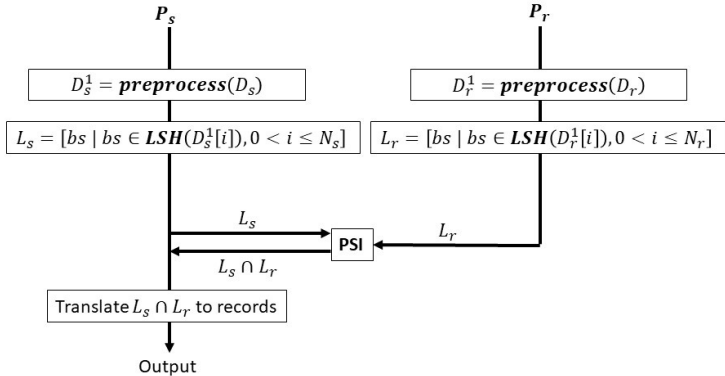– Our program is freely available for testing at [35].

Fig. 1: A high level illustration of our PPRL protocol. The parties P_s and P_r hold datasets D_s and D_r. They preprocess the data for every record and then feed the results into an LSH that outputs an ordered list of digest vectors L_s and L_r, respectively. These are fed into a PSI black box. Finally, P_s translates the PSI output to the matching record IDs.

1.1 Related work

To demonstrate our solution, we use a PSI instantiation that uses public-key cryptography; specifically, we use one that leverages the commutative properties of the Diffie-Hellmann (DH) key agreement scheme. This PSI construction was introduced in [20] with a similar construction even before that in [30]. Subsequent PSI works consider other, more complex cryptographic primitives such as homomorphic encryption (HE) [6] and oblivious transfer (OT) [32]. While the latter solutions may offer an interesting tradeoff in terms of performance and security, we decided to stick with the basic DH-style protocol due to its simplicity and the fact that its primitives were already standardized [2]. Because we use PSI as a blackbox, we can also benefit from most of the advantages that the other methods provide such as performance and security guarantees.

Our solution follows previous works in considering a *balanced* case, where the two datasets are roughly equal in size. An example, for a PSI over unbalanced sets was studied in [5]. In fact, there were attempts to use PSI for PPRL before

this paper. However, they were either noted to be inefficient [38] or relied on a different techniques such as term frequency–inverse document frequency (TF-IDF) [34], which is more appropriate for comparing documents, rather than short record fields (such as names or addresses). Furthermore, the protocol of [34] can only compare given record pairs. This implies the need for $\mathcal{O}(n^2)$ operations, in contrast to our method, which requires $\mathcal{O}(n)$ operations.

A complete survey of PPRL techniques and challenges is available at [18,38], in which we observed solutions that use different cryptographic primitives. For example, [14,39] relies on HE, which is known for its high computational cost. For example, [14] reports that it took somewhat less than two hours to evaluate $20,000$ patient records, which is less records than in our evaluations by several orders of magnitude. Other works [4,36] use garbled circuits, which can still be inefficient, while other multi-party computation solutions such as [26] can incur high communication costs [7]. Another example is the fuzzy volts approach, which uses secure polynomial interpolations [31], but only reports results for around $1,000$ records. Other solutions [19,33] overcome the leakage issue by using differential privacy, which anonymizes the data to maintain privacy. We see it as an orthogonal approach to ours.

Many PPRL works use Bloom filter encodings [37], which use a locality preserving hash (LPH) function over the data. The main advantage of the Bloom filter is speed. The difference between LPH and LSH is that LPH is data-dependent, i.e., for three records p, q, r, a metric d, and an LPH function l_p

$$d(p,q) < d(q,r) \implies d(l_p(p), l_p(q)) < d(l_p(q), l_p(r))$$

This relation complicates the evaluation of the protocol leakage. The lack of a formal analysis for Bloom filter based solutions caused several attacks on them [8,9,27,28]. A survey of attacks and countermeasures for this method can be found in [15]. Our solution's use of LSH has an advantage over Bloom filters as it is data-independent and more robust against the above attacks. A method that combines Bloom filters and LSH was presented in [16,24]. In contrast to this one, our solution only uses LSH, which simplifies the privacy analysis. Moreover, our use of PSI hides the LSH output and thus prevents offline attacks. In addition, [24] requires use of a third-party and demonstrates a solution that took more than an hour to match $300K$ records. Another recent example is [26], which runs in $\mathcal{O}(n \cdot polylog(n))$ and proved to be cryptographically secure in the semi-honest security model. However, the method analyzed $4,096$ records in 88 minutes and it is not clear whether this method can scale to handle more than $100K$ records.

Organization. The paper is organized as follows. Section 2 provides some background notation and describes the required preliminaries for this work. In Section 3 we present and discuss several possible definitions of PPRL protocols. We provide a high level description of our solution in Section 4 and provide further details about our implementation in Appendix D. We report our experimental setup and results in Section 5 and conclude in Section 6.

2   Preliminaries and notation

We denote the concatenation of two strings by $s_1 \mid s_2$. The function $Eq(s, r)$ returns 1 when two strings are equal and 0 otherwise. An ordered list of elements A is marked with square brackets, e.g., $A = [5, 3, 8]$ and we access its ith element by $A[i]$. A permutation π can either return a permuted list when operating on an ordered list, or the index of a permuted element within that list when the input is another index. For example, let $\pi : x \mapsto x + 1 \pmod 4$ be a permutation, then $\pi([5, 6, 7, 8]) = [8, 5, 6, 7]$, $\pi(2) = 3$, and $\pi(3) = 0$. Uniform random sampling from a set U is denoted by $u \xleftarrow{\$} U$.

2.1   Entity resolution (ER)

An ER method gets as input two datasets of N_s and N_r records from record spaces $\mathcal{R}$: $D_s = \{s_1, s_2, \ldots, s_{N_s}\}$ and $D_r = \{r_1, r_2, \ldots, r_{N_r}\}$, respectively. It evaluates the similarity of every two records using a similarity measure $\mu : \mathcal{R} \times \mathcal{R} \to [0, 1]$ and an associated similarity indicator

$$I_t^\mu : \mathcal{R} \times \mathcal{R} \longrightarrow \{0, 1\}$$

$$(s, r) \longmapsto \begin{cases} 1 & \mu(s, r) \geq t \\ 0 & otherwise \end{cases}$$

The ER method uses the similarity indicator to facilitate a bipartite graph $G = (U, V, E)$, where the nodes of U, V are the records of D_s, D_r, respectively, and for every two nodes ($u \in U$, $v \in V$), an edge exists in E if $I_t^\mu(u, v) = 1$.

PPRL. Informally, a PPRL protocol is an ER method executed by two parties: a sender P_s and a receiver P_r, who privately hold D_s and D_r, respectively. At the end of the protocol, P_r learns the similarity edges E while P_s learns nothing. We provide a formal definition in Section 3. Specifically, our PPRL solution uses the LSH and PSI primitives, described next.

2.2   Local sensitive hash (LSH)

An LSH [29] is a hash function that deliberately hashes *similar* inputs to the same output hash value. We are interested in the similarity of strings i.e., the content of the record fields. Therefore, we use the LSH from [29], which is based on the Jaccard index and on *Min-Hashes*, as demonstrated in Figure 2.

Jaccard index (a.k.a. the Jaccard similarity coefficient) is a similarity measure for strings. The procedure for computing the Jaccard index of two inputs strings (s, r) splits each normalized string into the set of all overlapping sub-strings of given lengths, termed *k-shingles* (or *k-grams*), where k is the length of the sub-strings. We use small letters to denote strings or the corresponding records, and capital letters to denote their associated sets of k-shingles. The Jaccard index for records s, r is

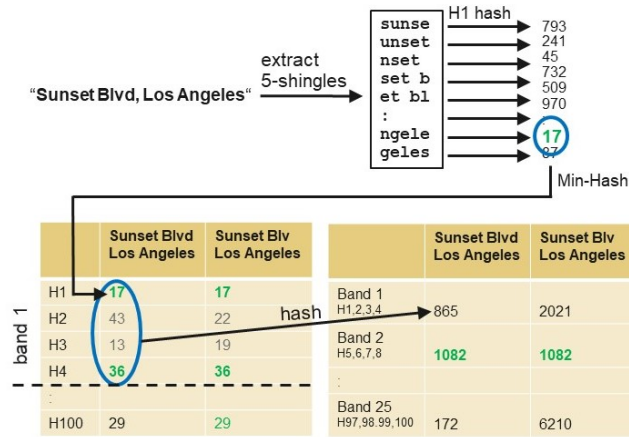$$J(s, r) = \frac{|S \cap R|}{|S \cup R|} \tag{1}$$

Fig. 2: Computing the LSH for a string: shingles are extracted from the normalized string, and then min-hashes are evaluated and grouped into bands that are hashed to a list of signatures.

when the context is clear we use J instead of $J(s,r)$.

Example 1. Consider the strings:

$$s = \text{`Sunset Blvd, Los Angeles'}$$
$$r = \text{`Sunet Blvd, Los Angeles'}$$

that are normalized into

$$\text{`sunset blvd los angeles'}$$
$$\text{`sunet blvd los angeles'}$$

and then split into the set of 19 and 18 shingles of length $k = 5$, respectively:

$$S = \{\text{`sunse', `unset', `nset ', `set b', ..., `ngele', `geles'}\}$$
$$R = \{\text{`sunet', `unet ', `net b', `et bl', ..., `ngele', `geles'}\}$$

Here, the Jaccard index is $J = \frac{15}{22} \approx 0.68$. Using longer shingles of length $k = 11$ would result in a lower Jaccard index of $J = 0.56$.

It is possible to instantiate a PPRL solution that relies on the Jaccard index. The drawback of such a protocol is that it has quadratic complexity in the size of the datasets. For linear complexity, we use Min-Hash.

Definition 1 (Min-Hash [29]). *For a collision-resistant hash function H with an integer output digest and an integer k, a Min-Hash function receives a string s as input, converts it to a k-shingles set S, and returns*

$$\mathtt{MinH}_k^H(s) = \min_{e \in S} H(e)$$

When the context is clear we write MinH instead of $MinH_k^H$.

Observation 1 **([29])** *For two normalized records s and r, a collision resistant hash function H, and $k > 0$, it follows that $Pr\left[MinH_k^H(s) = MinH_k^H(r)\right] = J(s,r)$.*

An LSH involves applying P different Min-Hash functions to a string s. The outputs are split into B *bands* of R digests ($P = BR$). The concatenation of the R digests of each band is again hashed to produce the *signature* of the band, where the same signature hash function is used for all bands. An LSH output is a tuple with these band signatures.

Definition 2 (LSH). *For $k, R, B \in \mathbb{N}$, $P = RB$, distinct collision-resistant hash functions H_i, $1 \le i \le P$ and another collision-resistant hash functions G, a band b_j, $1 \le j \le B$ over a string s is the concatenation*

$$b_j(s) = \texttt{MinH}_k^{H_{R\cdot(j-1)+1}}(s) \mid \cdots \mid \texttt{MinH}_k^{H_{R\cdot(j-1)+R}}(s)$$

and the LSH output over a string s is the ordered list

$$LSH(s) = \left[G\left(b_1(s)\right), G\left(b_2(s)\right), \ldots, G\left(b_B(s)\right)\right]$$

Two LSH tuples are considered to be a match if they share at least one common signature. We denote this by the indicator function

$$\texttt{LSHMatch} : \mathcal{R} \times \mathcal{R} \longrightarrow \{0,1\}$$

$$(s,r) \longmapsto \begin{cases} 1 & 1 \le \sum\limits_{i=1}^{B} Eq\left(LSH(s)[i], LSH(r)[i]\right) \\ 0 & otherwise \end{cases}$$

Observation 2 **([29])** *For two records s, r,*

$$Pr[\texttt{LSHMatch}(s,r) = 1] = 1 - (1 - J^R)^B \qquad (2)$$

Example 2. Figure 2 demonstrates an LSH with $P = 100$, $B = 25$, $R = 4$, where $\texttt{MinH}_5^{H_1}(s_1) = 17$, $\texttt{MinH}_5^{H_2}(s_1) = 43$, etc. Subsequently, every sequence of R digests is concatenated and hashed to produce a band signature, with a total of B band signatures, which form the LSH of s_1, $LSH(s_1) = (865, 1082, \ldots, 172)$. Repeating the process for s_2, we observe a match in the signature of the second band for the two compared strings; this means that the two LSHs match and the strings match with a high probability.

2.3   Private set intersection (PSI)

PSI is a cryptographic protocol that allows two parties to compute the intersection of their private sets without revealing anything beyond this fact or beyond the size of the intersected sets to the other party. PSI is a special case of PPRL,

which considers only exact matches. Some variations of PSI allow the parties to learn just the cardinality of the intersection.

Many PSI solutions exist (see Section 1). In this work, we use a unidirectional variant of the DH-PSI [30], as presented in Figure 3. The two parties P_s and P_r first agree on a group $\mathbb{G}$ and a collision-resistant hash function H, and each party generates its own secret key sk_s and sk_r, respectively. Subsequently, both parties hash and encrypt their records using their private keys and send them to the other party. In addition, P_r encrypts the output of P_s using its secret key and sends the results back to P_s. Finally, P_s learns the intersection of the two datasets.

$$\underline{P_s\ (sk_s)} \hspace{6cm} \underline{P_r\ (\ sk_r)}$$

$$D'_s = \{H(s)^{sk_s}|s \in D_s\}$$

$$\xrightarrow{\hspace{3cm} D'_s \hspace{3cm}}$$

$$D'_r = \{H(r)^{sk_r}|r \in D_r\}$$
$$D''_s = \{(s')^{sk_r}|s' \in D'_s\}$$

$$\xleftarrow{\hspace{2cm} D'_r\ \text{and}\ D''_s \hspace{2cm}}$$

$$D''_r = \{(r')^{sk_s}|r' \in D'_r\}$$
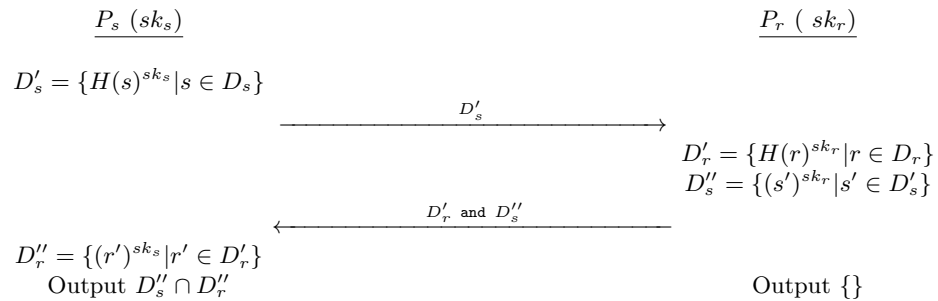$$\text{Output } D''_s \cap D''_r \hspace{4cm} \text{Output } \{\}$$

Fig. 3: One side DH-PSI

Informally, the security of these protocols against semi-honest adversaries is guaranteed by the one-way property of the hash function, the computational hardness of the decisional DH, and the one-more-DH [17] assumptions (see definitions in Appendix A). The decisional DH is used to hide the data in transit from eavesdroppers, while the one-more-DH assumption is used to prevent P_s from generating new records in the name of P_r.

One DH-PSI variant is the mutual DH-PSI, which includes one extra round: P_s sends D''_r to P_r so that P_r can also compute the intersection. However, here an eavesdropper learns both D''_s and D''_r and can therefore learn the cardinality of the intersection $D''_s \cap D''_r$.

One issue with DH-PSI is that it is susceptible to man-in-the-middle attacks [13]. To mitigate this attack and the leakage of the mutual DH-PSI's intersection cardinality,we assume that the transportation is encrypted and authenticated using TLS 1.3.

3 PPRL

PPRL is an ER protocol between two parties P_s and P_r, with private datasets D_s and D_r of sizes N_s and N_r, respectively; these records have a similarity measure $\mu(\cdot, \cdot)$, and some additional privacy requirements. These requirements may lead to several security models and several formal definitions of PPRL.

The most intuitive way to define privacy for PPRL is by following the PSI privacy notion: P_s only learns N_r and the intersection $D_s \cap D_r$, i.e., all records that exactly match in all fields while P_r only learns N_s. Note that in both PSI and PPRL, P_s and P_r need to share the nature of the information contained in their datasets with each other to decide which QIDs they can validly compare.

The difference between PSI and PPRL is that PSI only returns exact matches according to some uniquely identifying QIDs, while PPRL returns matching records up to some similarity indicator and according to non-unique QIDs. For example, a PSI protocol may rely on users' SSNs, while a PPRL protocol may compare first and last names. Thus, a PPRL may inadvertently match "David Doe" with "Davy Don" even if they represent different entities (users).

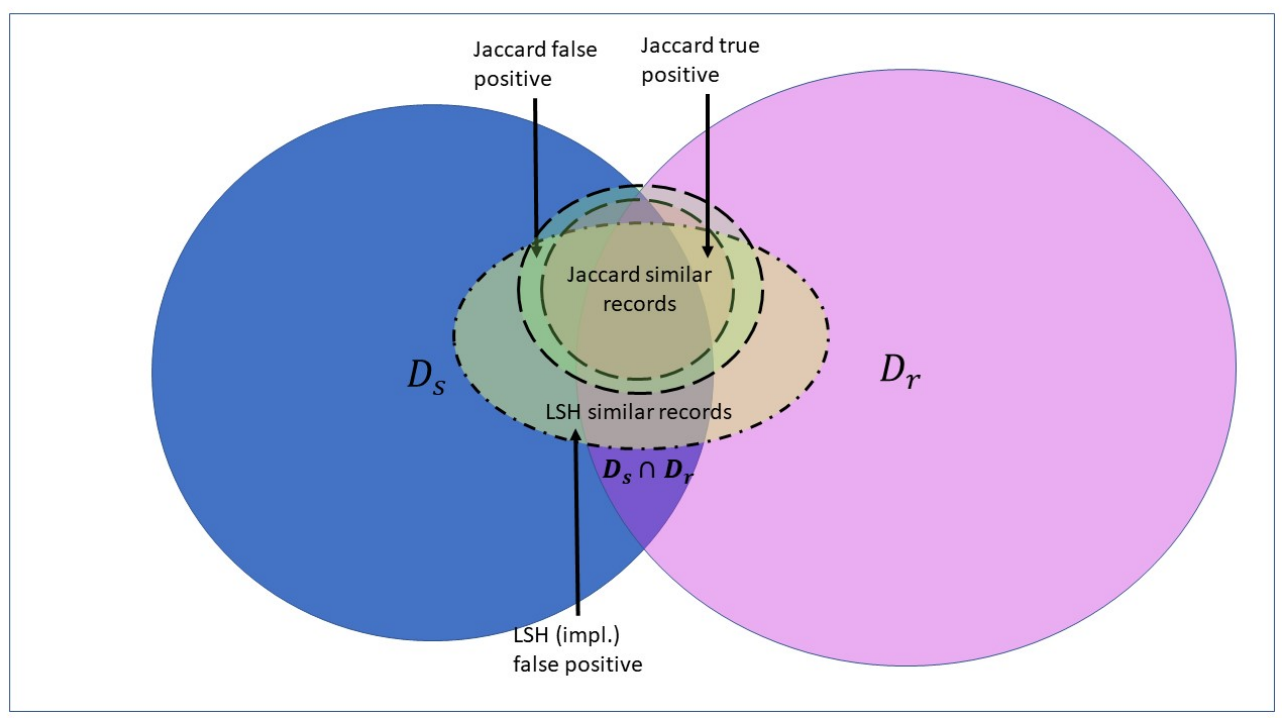

Fig. 4: A Venn diagram of different ER outputs applied on two datasets D_s and D_r. The ER methods are: matching only identical pairs of records (purple), matching pairs of records with a Jaccard index above some threshold (green), and matching pairs of records with matching LSH indicators (yellow).

Figure 4 shows a Venn diagram for the output of different ER solutions on D_s and D_r datasets. With the exact matching method ($D_s \cap D_r$) no privacy risks occur since it only reveals the agreed-upon intersection[3]. In contrast, when using the Jaccard similarity to compute the matches, the parties learn: a) records in $D_s \cap D_r$, which is ok; b) records outside $D_s \cap D_r$ that represent the same entity (true-positive), which is also ok; c) records outside $D_s \cap D_r$ that represent different entities (false-positive), which may break the privacy of the parties. In general, any PPRL protocol must assume this kind of leakage, and should do its best to quantify it, e.g., by assuming the existence of a bound τ on the similarity false-positive rate.

[3] In practice, if P_s learns that both parties share a record with the same SSN and at a later stage learns that the other record fields do not match, then it may deduce that D_r contains a record with a very close SSN that leaks information. Following previous studies, we only consider leaks that occur as a result of the protocol itself.

Definition 3 (PPRL). *A PPRL protocol $\mathcal{P}$ between two parties P_s, P_r with datasets D_s, D_r, respectively, a similarity measure μ, a measure indicator I_t^μ for a fixed threshold t with a false-positive rate bounded by τ, has the following properties.*

– **Correctness:** *$\mathcal{P}$ is correct if it outputs to P_s the set*

$$res = \{(s, Enc(r)) \mid s \in D_s, r \in D_r, I_t^\mu(s, r) = 1\},$$

where $Enc(r)$ is an encryption of r under a secret key of P_r.
– **Privacy:** *$\mathcal{P}$ maintains privacy if P_s only learns res and N_r, and P_r only learns N_s.*

Corollary 1. *The leaked information of P_r in $\mathcal{P}$ is bounded by $\tau \cdot \frac{|res|}{N_r}$.*

Definition 3 assumes the existence of τ but only implicitly uses it. The reason is that τ does not always exist. In many cases, it can be empirically estimated based on prior data or based on perturbed synthetic data. However, relying solely on empirical estimates increases the ambiguity of the privacy definition for such protocols. Moreover, in many cases, τ depends on data from the two datasets that have different distributions, which none of the parties know in advance. Another reason for only implicitly relying on τ is that the leaked information in Cor. 1 depends on res and can only be computed after running the protocol.

While τ bounds the privacy leak from above, there is still the issue of quantifying the exact leakage after the protocol ends. It is not clear how the parties can verify the number of false-positive cases without revealing private data. Usually, an ER protocol is used when the compared records do not include uniquely identifying fields (such as an SSN) and thus the parties cannot compute the exact matches using PSI. Consequently, their only way to verify matches is by revealing their private data. To assist in this task, we define a protocol called a revealing PPRL.

Definition 4 (Revealing PPRL). *A revealing PPRL protocol $\mathcal{P}$ is a PPRL protocol $\mathcal{P}'$, where P_r also learns $u = \{Enc(r) \mid (s, Enc(r)) \in \mathcal{P}'.res\}$ and P_s also learns*

$$res' = \{(r, Enc(r)) \mid (s, Enc(r)) \in \mathcal{P}'.res\},$$

In words, P_r learns which of its own records are matched, and P_s learns the field content of the matched records of the other party. The simplest way to achieve a revealing PPRL is for P_s to send u to P_r, who will then decrypt its values and hand them back to P_s. The difference between Definitions 3 and 4 is that in the latter, P_s learns the values of P_r's records instead of just their encryption. While this definition leaks more data from P_r to P_s, it is easier to analyze because now P_s can verify the matches with some probability and learn the estimated number of false-positives. We also consider the definitions of the associated mutual PPRL and the mutual revealing PPRL.

Definition 5 (Mutual PPRL). *A PPRL protocol* $\mathcal{P}$ *between two parties* P_s, P_r *with datasets* D_s, D_r, *respectively, a similarity measure* μ, *a measure indicator* I_t^{μ} *for a fixed threshold t with a false-positive rate bounded by* τ, *has the following properties.*

- **Correctness:** $\mathcal{P}$ *is correct if it outputs* res_s *(resp.* res_r*) to* P_s *(resp.* P_r*), where*
$$res_s = \{(s, Enc(r)) \mid s \in D_s, r \in D_r, I_t^{\mu}(s,r) = 1\}$$
$$res_r = \{(r, Enc(s)) \mid s \in D_s, r \in D_r, I_t^{\mu}(s,r) = 1\},$$
and $Enc(r)$ *(resp.* $Enc(s)$*) is an encryption of* r *(resp.* s*) under a secret key of* P_r *(resp.* P_s*).*
- **Privacy:** $\mathcal{P}$ *maintains privacy if* P_s *only learns* res_s *and* N_r, *and* P_r *only learns* res_r *and* N_s.

The mutual revealing PPRL is similarly defined. The difference between the mutual PPRL and the revealing PPRL in terms of privacy is that in the mutual PPRL, P_r can match the encryption of P_s records to its records and therefore gains more information while P_s only learns the encryption of P_r records.

In the PPRL protocols described above, the two parties learn the intersection of their datasets. However, in some scenarios, the parties merely need to learn the *number* of matches and do not wish to reveal the identity of the matched records to the other party. To this end, we define an N-PPRL protocol.

Definition 6 (N-PPRL). *A PPRL protocol* $\mathcal{P}$ *between two parties* P_s, P_r *with datasets* D_s, D_r, *respectively, a similarity measure* μ, *a measure indicator* I_t^{μ} *for a fixed threshold t with a false-positive rate bounded by* τ, *has the following properties.*

- **Correctness:** $\mathcal{P}$ *is correct if it outputs to* P_s *the value*
$$N_{s \cap r} = |\{(s,r) \mid s \in D_s, r \in D_r, I_t^{\mu}(s,r) = 1\}|,$$
- **Privacy:** $\mathcal{P}$ *maintains privacy if* P_s, *(resp.* P_r*) only learns* $N_{s \cap r}, N_r$ *(resp.* N_s*).*

The mutual N-PPRL protocol is similarly defined.

4 Our solution

Our PPRL solution (hereafter: LSH-PSI PPRL) is an ER protocol that uses `LSHMatch` as its similarity indicator, where for privacy reasons, the parties cannot directly share the LSH results. The reason depends on whether the LSH is a preimage-resistant hash function or not. When it is not, P_r and P_s can simply inverse the LSH results for records that are not in the intersection and reveal private information of P_s, P_r, respectively. But even when it is, the solution's privacy depends on the LSH input entropy, where the parties can maintain an offline brute force attack against the LSH records of the other party.
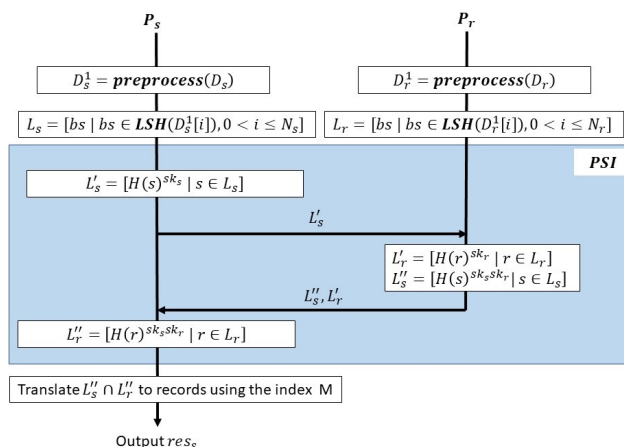
Fig. 5: Schematic of the LSH-PSI PPRL protocol.

To mitigate the privacy issue, we use a PSI protocol. The two parties first compute the LSH band signatures of all their records and then apply a PSI protocol over these signatures. Finally, P_s maps back the intersected signatures to the original records to learn the set of similar records. The concrete properties of LSHMatch can be tuned using the B and R LSH parameters. Figure 6 presents the LSH-PSI protocol, and Figure 5 illustrates it schematically.

Our protocol is defined against semi-honest (honest-but-curious) adversaries, where all parties do not deviate from the protocol, and their inputs are genuine. Nevertheless, they may record and analyze all the intermediate computations and messages from the other parties to get more information.

Remark 1. The two parties must use the same preprocessing techniques to increase the efficiency of the underlying ER method. In addition, the LSH-PSI protocol assumes that the pre-processing phase runs some deduplication protocol on the dataset of every party. Otherwise, P_s can extract information from pairs of matching records $s_1, s_2 \in D_s$, where s_1 matches a record in D_r but s_2 does not.

Remark 2. The PSI protocol is executed for all records at once and not per record, therefore it is critical to preserve the order of the signatures exchanged between the parties, i.e., of L_s' and L_s'' in Figure 5. Otherwise, it will be impossible to match the records in Step 4 of Figure 6. In Section 4.1, we discuss the case where P_r does *not* preserve the order of P_s encrypted signatures.

The purpose of using the permutation π_p in Step 1b is to avoid the case where the other party learns information about "missing" records. For example, suppose that the records in D_r are ordered alphabetically according to a first name QID, and that P_s learns that *Jerry* and *Joseph* are in the intersection. If *Jerry* and *Joseph* happen to belong to adjacent records in D_r, then an honest
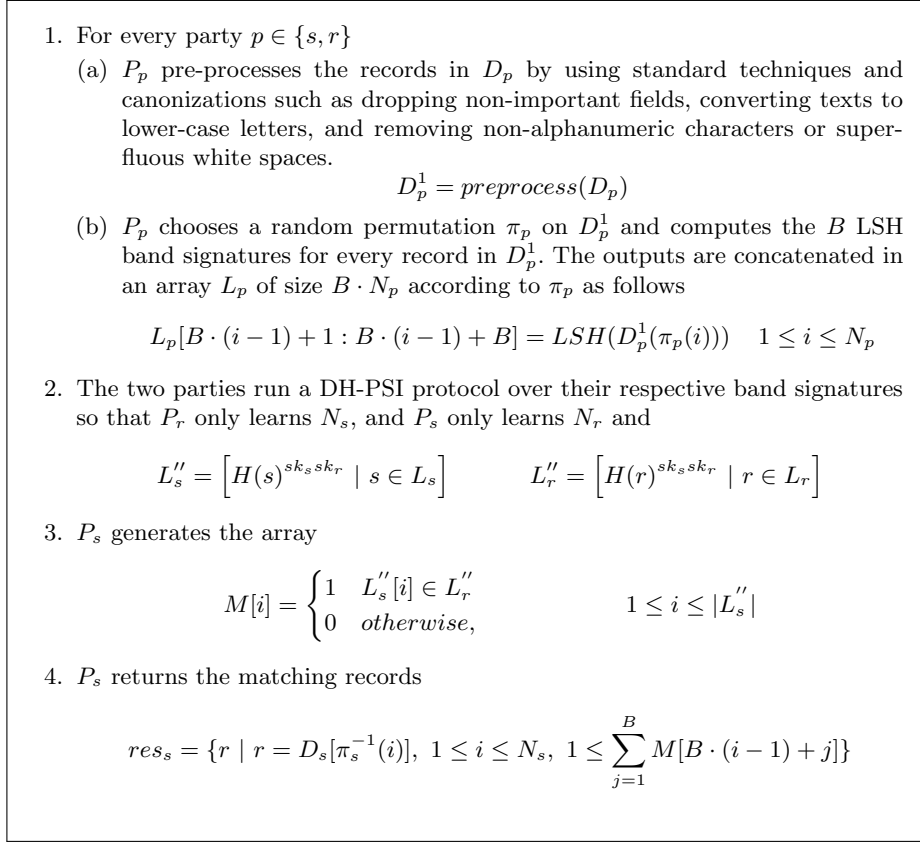
1. For every party $p \in \{s, r\}$

    (a) P_p pre-processes the records in D_p by using standard techniques and canonizations such as dropping non-important fields, converting texts to lower-case letters, and removing non-alphanumeric characters or superfluous white spaces.
    $$D_p^1 = preprocess(D_p)$$

    (b) P_p chooses a random permutation π_p on D_p^1 and computes the B LSH band signatures for every record in D_p^1. The outputs are concatenated in an array L_p of size $B \cdot N_p$ according to π_p as follows
    $$L_p[B \cdot (i-1) + 1 : B \cdot (i-1) + B] = LSH(D_p^1(\pi_p(i))) \quad 1 \leq i \leq N_p$$

2. The two parties run a DH-PSI protocol over their respective band signatures so that P_r only learns N_s, and P_s only learns N_r and
$$L_s'' = \left[H(s)^{sk_s sk_r} \mid s \in L_s \right] \qquad L_r'' = \left[H(r)^{sk_s sk_r} \mid r \in L_r \right]$$

3. P_s generates the array
$$M[i] = \begin{cases} 1 & L_s''[i] \in L_r'' \\ 0 & otherwise, \end{cases} \qquad 1 \leq i \leq |L_s''|$$

4. P_s returns the matching records
$$res_s = \{r \mid r = D_s[\pi_s^{-1}(i)], \ 1 \leq i \leq N_s, \ 1 \leq \sum_{j=1}^{B} M[B \cdot (i-1) + j]\}$$

Fig. 6: The LSH-PSI PPRL protocol.

but curious P_s learns that P_r has no record for *John*. When using a permutation, the only way for P_s to deduce the same information is by learning all the records in D_r. A concrete example of Steps 1.b - 3 is given in Appendix B.

Theorem 1. *The LSH-PSI PPRL protocol is a PPRL protocol according to Definition 3 where the similarity indicator is* `LSHMatch`*. This protocol is secure against semi-honest adversaries.*

Proof. **Correctness.** The correctness of the protocol follows from the fact that the intersection $L_s \cap L_r$ has a one-to-one correlation with the encrypted band signatures $L_s'' \cap L_r''$.

    Privacy of P_s. By the discrete-log assumption, P_r only gets to see N_s elements that are indistinguishable from random values. Thus, P_r only learns N_s.

    Privacy of P_r. P_s gets from P_r the values of L_s' and L_r raised to the power of P_r's secret key. By the discrete-log assumption, these values are indistinguishable from random to P_s. Except that P_s can raise L_r' values to the power of

its own secret key and then intersect the results with L_s''. This intersection of random values is used by P_s to identify matching signatures, which is expected by Definition 3. Because P_s learns nothing from values outside the intersection, we say that it only learns res and N_r as expected.   □

Remark 3. Similar to the DH-PSI case, the use of TLS 1.3 allows the parties to mutually authenticate themselves and to avoid the attack presented in [13]. Still, as a defense-in-depth mechanism, the parties in every PPRL session should avoid reusing secret keys to avoid man-in-the-middle attacks.

4.1   PPRL variants

Based on the above protocol, we construct three other protocols: a mutual PPRL protocol, an N-PPRL protocol, and a revealing PPRL protocol, where the latter immediately follows the definition.

A mutual PPRL protocol. To establish a mutual PPRL protocol, we modify Step 2 of Figure 6 to use the mutual DH-PSI protocol of Section 2.3. The security of the protocol follows from either the security of the mutual DH-PSI, or from the fact that the mutual protocol is equivalent to running the original PPRL protocol twice: first between P_s and P_r, and subsequently between P_r and P_s. Note that P_s cannot reduce the communication by sending only records that are in the intersection because then an eavesdropper can learn the intersection size. This claim is valid even when using a secure communication channel (e.g., TLS 1.3).

An N-PPRL protocol. To achieve an N-PPRL protocol, we could have simply counted the number of elements in the intersection set res, but this would reveal to P_s more information beyond $N_{s \cap r}$. Instead, we suggest reordering the encrypted band signatures during the DH-PSI in a way that hides the identity of the matched records but still enables them to be counted. Specifically, we ask P_r to apply a secret permutation to L_s'' before sending it to P_s. This permutation has a special property that permutes together the groups of adjacent B signatures that originate from the same record, otherwise, P_s will not be able to distinguish between the cases

1. $|LSH(s_1) \cap LSH(r_1)| = 1$ and $|LSH(s_2) \cap LSH(r_2)| = 1$
2. $|LSH(s_1) \cap LSH(r_1)| = 2$ and $|LSH(s_2) \cap LSH(r_2)| = 0$

We call the above permutation an intra-permutation of records. In addition, we apply an inter-permutation of records, where we separately permute the B signatures in each group of signatures in L_s'' that originate from the same record.

5   Experiments

Experimental setup. We carried out the experiments on two machines that are located in different local area networks (LANs). We measured an average of 65 ms round-trip latency between them.

Table 1: Accuracy of our PPRL protocol over the NCVR snapshots.

Set size	FN	FP	TP	Precision (%)	Recall (%)	F1 (%)
10^4	19	21	653	96.88	97.17	97.03
10^5	1,369	1,665	55,682	97.1	97.6	97.35
10^6	22,233	19,365	847,724	97.77	97.44	97.61

– Machine A has an Intel® Xeon® CPU E5-2620 v3 @ 2.40GHz, with 12 physical cores and 377 GB of RAM.
– Machine B has an Intel® Xeon® CPU E5-2699 v4 @ 2.20GHz, with 44 physical cores and 744 GB of RAM.

We set machine A to run P_s and machine B to run P_r with $N_s \approx N_r$.

Our code is written in C++ and runs on Ubuntu 20.04. It uses OpenSSL version 1.1.1f to establish secure TLS 1.3 connections between the two parties. In addition, it uses OpenSSL hash function implementation (concretely, H=SHA256) and DH operations (concretely, elliptic curve DH operations over the NIST P-256 curve). We report communications in KB and running time in seconds. We also provide a breakdown of the different running time phases: communication and computations per party. For the measurements, we separated the communication phases from the computation phases, which in a real scenario can be pipelined to run in parallel.

For the evaluations, we considered two dataset cases: a) The North Carolina voter register (NCVR) dataset[4], which is commonly used for PPRL evaluations; b) a synthetic dataset that we generated and made available in [35].

NCVR datasets. We used the November 2014 and November 2017 snapshots of the NCVR datasets. Prior to running the PPRL protocol, we deduplicated the snapshots by eliminating duplicate records with identical "NCID" or with identical values in the 'first_name', 'last_name', 'midl_name', 'birth_place' and 'age' fields. Subsequently, we removed the NCID field from the two snapshots, and ran our PPRL protocol on the two snapshots. A reported matching pair was considered to be a true-positive event if the two reported records share the same NCID value. Table 1 shows the accuracy breakdown of the LSH we used by reporting the number of false-negative (FN), false-positive (FP), and true-positive (TP) events, together with the precision, recall, and F1 results when sampling sets of fixed sizes from the above snapshots. Note that while the precision is high, the absolute number of false-positives may be regarded as too high for some users. See Section C.1 for ways to tune the process and balance the number of false positive and false negative cases while considering the protocol performance.

Synthetic dataset. We generated two synthetic datasets using IBM InfoSphere® Optim^{TM} Test Data Fabrication [21] with the following fields: 'first name', 'last name', 'email', 'email domain', 'address number', 'address location', 'address

---

[4] https://www.ncsbe.gov/results-data/voter-registration-data,lastaccessedMar2022.

16

Table 2: Performance results on the synthetic dataset for different samples of the original dataset.

Set sizes	Comm. (KB)	Comm. time (s)	Offline time (s)	Total time (s)
2^8	$5.68 \cdot 10^3$	3	1	4
2^{12}	$9.04 \cdot 10^4$	17	2	19
2^{16}	$1.44 \cdot 10^6$	237	36	273
2^{20}	$1.19 \cdot 10^7$	1,959	608	2,567

line', 'city', 'state', 'country', 'zip base', 'zip ext', 'phone area code', 'phone exchange code' and 'phone line number', where $N_s \approx N_r \approx 1,000,000$. We generated the datasets in a way that only 100 records in the two datasets represent identical entities. The pairs of records that describe these shared entities sometimes have identical fields and sometimes fields with minor typos, different styles, and other types of minor differences, which are still small enough to warrant the assumption that the similar records in fact describe the same entity. Our PPRL protocol identified all the matching records. The performance evaluation of the protocol is given in Table 2.

6 Conclusion

We presented a novel PPRL solution that relies on LSH to identify similar records while using PSI to ensure privacy. We formally defined the privacy guarantees that such a protocol provides and evaluated its efficiency. Our results show that it takes $11 - 45$ minutes (depending on the network settings) to perform a PPRL solution comparing two large datasets with 2^{20} records per dataset. Note that none of the results presented in Section 1.1 reported comparable speeds for such large datasets. This makes our solution practical and attractive for companies and organizations. We made our implementation available for testing at [35].

We proposed a PPRL framework that can use different PSI protocols as long as they provide the same security guarantees defined above. We demonstrated our solution using an ECDH PSI protocol. It may be an interesting direction to implement and test the protocol using other solutions that can further improve its performance and overall bandwidth.

References

1. Baker, D.B., Knoppers, B.M., Phillips, M., van Enckevort, D., Kaufmann, P., Lochmuller, H., Taruscio, D.: Privacy-Preserving Linkage of Genomic and Clinical Data Sets. IEEE/ACM Transactions on Computational Biology and Bioinformatics **16**(4), 1342–1348 (2019). https://doi.org/10.1109/TCBB.2018.2855125
2. Barker, E., Chen, L., Moody, D.: Recommendation for Pair-Wise Key- Establishment Schemes Using Integer Factorization Cryptography (Revision 1) (2014). https://doi.org/10.6028/NIST.SP.800-56Br1

3. Carter, J.L., Wegman, M.N.: Universal classes of hash functions. Journal of computer and system sciences **18**(2), 143–154 (1979)

4. Chen, F., Jiang, X., Wang, S., Schilling, L.M., Meeker, D., Ong, T., Matheny, M.E., Doctor, J.N., Ohno-Machado, L., Vaidya, J.: Perfectly secure and efficient two-party electronic-health-record linkage. IEEE Internet Computing **22**(2), 32–41 (2018). https://doi.org/10.1109/MIC.2018.112102542

5. Chen, H., Huang, Z., Laine, K., Rindal, P.: Labeled psi from fully homomorphic encryption with malicious security. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. p. 1223–1237. CCS '18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3243734.3243836

6. Chen, H., Laine, K., Rindal, P.: Fast private set intersection from homomorphic encryption. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. p. 1243–1255. CCS '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3133956.3134061

7. Chen, Y.: Current approaches and challenges for the two-party privacy-preserving record linkage (pprl). Collaborative Technologies and Data Science in Artificial Intelligence Applications pp. 108–116 (2020), https://codassca2020.aua.am/wp-content/uploads/2020/09/2020_Codassca_Chen.pdf

8. Christen, P., Ranbaduge, T., Vatsalan, D., Schnell, R.: Precise and Fast Cryptanalysis for Bloom Filter Based Privacy-Preserving Record Linkage. IEEE Transactions on Knowledge and Data Engineering **31**(11), 2164–2177 (2019). https://doi.org/10.1109/TKDE.2018.2874004

9. Christen, P., Schnell, R., Vatsalan, D., Ranbaduge, T.: Efficient Cryptanalysis of Bloom Filters for Privacy-Preserving Record Linkage. In: Kim, J., Shim, K., Cao, L., Lee, J.G., Lin, X., Moon, Y.S. (eds.) Advances in Knowledge Discovery and Data Mining. pp. 628–640. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-57454-7_49

10. Churches, T., Christen, P.: Blind data linkage using n-gram similarity comparisons. In: Dai, H., Srikant, R., Zhang, C. (eds.) Advances in Knowledge Discovery and Data Mining. pp. 121–126. Springer Berlin Heidelberg, Berlin, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24775-3_15

11. Clifton, C., Kantarcioundefinedlu, M., Doan, A., Schadow, G., Vaidya, J., Elmagarmid, A., Suciu, D.: Privacy-preserving data integration and sharing. In: Proceedings of the 9th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery. p. 19–26. DMKD '04, Association for Computing Machinery, New York, NY, USA (2004). https://doi.org/10.1145/1008694.1008698

12. Cong, K., Moreno, R.C., da Gama, M.B., Dai, W., Iliashenko, I., Laine, K., Rosenberg, M.: Labeled psi from homomorphic encryption with reduced computation and communication. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. p. 1135–1150. CCS '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3460120.3484760

13. Cui, H., Yu, Y.: A not-so-trival replay attack against dh-psi. Cryptology ePrint Archive, Report 2020/901 (2020), https://ia.cr/2020/901

14. Essex, A.: Secure Approximate String Matching for Privacy-Preserving Record Linkage. IEEE Transactions on Information Forensics and Security **14**(10) (2019). https://doi.org/10.1109/TIFS.2019.2903651

15. Franke, M., Rahm, E.: Evaluation of Hardening Techniques for Privacy-Preserving Record Linkage (2021)

16. Franke, M., Sehili, Z., Rahm, E.: Parallel Privacy-Preserving Record Linkage using LSH-based blocking. International Conference on Internet of Things, Big Data and Security (IoTBDS) (2018), https://www.scitepress.org/Papers/2018/66827/66827.pdf

17. Freeman, D.: Pairing-based identification schemes. Cryptology ePrint Archive, Report 2005/336 (2005), https://ia.cr/2005/336

18. Gkoulalas-Divanis, A., Vatsalan, D., Karapiperis, D., Kantarcioglu, M.: Modern privacy-preserving record linkage techniques: An overview. IEEE Transactions on Information Forensics and Security **16**, 4966–4987 (2021). https://doi.org/10.1109/TIFS.2021.3114026

19. He, X., Machanavajjhala, A., Flynn, C., Srivastava, D.: Composing Differential Privacy and Secure Computation: A Case Study on Scaling Private Record Linkage. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1389–1406. CCS '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3133956.3134030

20. Huberman, B.A., Franklin, M., Hogg, T.: Enhancing privacy and trust in electronic communities. In: Proceedings of the 1st ACM Conference on Electronic Commerce. p. 78–86. EC '99, Association for Computing Machinery, NY, USA (1999). https://doi.org/10.1145/336992.337012

21. IBM: Ibm infosphere® optim™ test data fabrication (2022), https://www.ibm.com/products/infosphere-optim-test-data-fabrication

22. Ioffe, S.: Improved consistent sampling, weighted minhash and l1 sketching. 2010 IEEE International Conference on Data Mining pp. 246–255 (2010)

23. Karapiperis, D., Gkoulalas-Divanis, A., Verykios, V.S.: FEDERAL: A Framework for Distance-Aware Privacy-Preserving Record Linkage. IEEE Transactions on Knowledge and Data Engineering **30**(2), 292–304 (2018). https://doi.org/10.1109/TKDE.2017.2761759

24. Karapiperis, D., Verykios, V.S.: A distributed near-optimal LSH-based framework for privacy-preserving record linkage. Computer Science and Information Systems **11**(2), 745–763 (2014). https://doi.org/10.2298/CSIS140215040K

25. Kargupta, H., Datta, S., Wang, Q., Sivakumar, K.: Random-data perturbation techniques and privacy-preserving data mining. Knowledge and Information Systems **7**(4), 387–414 (May 2005). https://doi.org/10.1007/s10115-004-0173-6

26. Khurram, B., Kerschbaum, F.: SFour: A Protocol for Cryptographically Secure Record Linkage at Scale. In: 2020 IEEE 36th International Conference on Data Engineering (ICDE). pp. 277–288 (2020). https://doi.org/10.1109/ICDE48307.2020.00031

27. Kroll, M., Steinmetzer, S.: Automated cryptanalysis of bloom filter encryptions of health records. In: Proceedings of the International Joint Conference on Biomedical Engineering Systems and Technologies - Volume 5. p. 5–13. BIOSTEC 2015, SCITEPRESS - Science and Technology Publications, Lda, Setubal, PRT (2015). https://doi.org/10.5220/0005176000050013

28. Kuzu, M., Kantarcioglu, M., Durham, E., Malin, B.: A constraint satisfaction cryptanalysis of bloom filters in private record linkage. In: Proceedings of the 11th International Conference on Privacy Enhancing Technologies. p. 226–245. PETS'11, Springer-Verlag, Berlin, Heidelberg (2011)

29. Leskovec, J., Rajaraman, A., Ullman, J.D.: Finding similar items. Mining of massive datasets pp. 73–130 (2014), http://infolab.stanford.edu/~ullman/mmds/ch3a.pdf

30. Meadows, C.: A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In: 1986 IEEE Symposium on Security and Privacy. pp. 134–134 (1986). https://doi.org/10.1109/SP.1986.10022
31. Mullaymeri, X., Karakasidis, A.: A Two-Party Private String Matching Fuzzy Vault Scheme. In: Proceedings of the 36th Annual ACM Symposium on Applied Computing. pp. 340–343. Association for Computing Machinery, NY, USA (2021). https://doi.org/10.1145/3412841.3442079
32. Pinkas, B., Schneider, T., Zohner, M.: Faster private set intersection based on OT extension. In: 23rd USENIX Security Symposium (USENIX Security 14). pp. 797–812. USENIX Association, San Diego, CA (Aug 2014), https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/pinkas
33. Rao, F.Y., Cao, J., Bertino, E., Kantarcioglu, M.: Hybrid Private Record Linkage: Separating Differentially Private Synopses from Matching Records. ACM Trans. Priv. Secur. **22**(3) (apr 2019). https://doi.org/10.1145/3318462
34. Ravikumar, P., Cohen, W.W., Fienberg, S.E.: A secure protocol for computing string distance metrics. PSDM held at ICDM (2004), https://www.cs.cmu.edu/afs/cs.cmu.edu/Web/People/wcohen/postscript/psdm-2004.pdf
35. Research, I.: Helayers (2022), https://hub.docker.com/r/ibmcom/helayers-pylab
36. Saleem, A., Khan, A., Shahid, F., Masoom Alam, M., Khan, M.K.: Recent advancements in garbled computing: How far have we come towards achieving secure, efficient and reusable garbled circuits. Journal of Network and Computer Applications **108**(January), 1–19 (2018). https://doi.org/10.1016/j.jnca.2018.02.006
37. Schnell, R., Bachteler, T., Reiher, J.: Privacy-preserving record linkage using Bloom filters. BMC Medical Informatics and Decision Making **9**(1), 41 (2009). https://doi.org/10.1186/1472-6947-9-41
38. Vatsalan, D., Sehili, Z., Christen, P., Rahm, E.: Privacy-Preserving Record Linkage for Big Data: Current Approaches and Research Challenges. In: Zomaya, A.Y., Sakr, S. (eds.) Handbook of Big Data Technologies, pp. 851–895. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-49340-4_25
39. Wong, K.S.S., Kim, M.H.: Privacy-preserving similarity coefficients for binary data. Computers and Mathematics with Applications **65**(9), 1280–1290 (2013). https://doi.org/10.1016/j.camwa.2012.02.028

A   Security assumptions

Definition 7 (Decisional DH (DDH)). *For a cyclic group $\mathbb{G}$, a generator g, and integers $a, b, c \in \mathbb{Z}$, the decisional DH problem is hard, if for every probabilistic polynomial-time (PPT) adversary $\mathcal{A}$*

$$|Pr[A(g, g^a, g^b, g^{ab}] = 1) -$$
$$Pr[A(g, g^a, g^b, g^c) = 1]| < negl(),$$

where the probability is taken over (g, a, b, c).

Definition 8 (Computational DH (CDH)). *For a cyclic group $\mathbb{G}$, a generator g, and integers $a, b \in \mathbb{Z}$, the computational DH problem is hard, if for every PPT adversary $\mathcal{A}$*

$$Pr[A(g, g^a, g^b] = g^{ab}) < negl(),$$

where the probability is taken over (g, a, b).

Definition 9 (One-more-DH (OMDH) [17]). *Let $\mathbb{G}$ be a cyclic group. The one-more-DH problem is hard, if for every PPT adversary $\mathcal{A}$ that gets a generator $g \in \mathbb{G}$ together with some power g^a and who has access to two oracles: $h^a = CDH_{g,g^a}(h)$ for some $h \in \mathbb{G}$, and $r \xleftarrow{\$} C()$ a challenge oracle that returns a random challenge point $r \in G$ and can only be invoked after all calls to the CDH_{g,g^a}, it follows that*

$$Pr[A(g, g^a, r \leftarrow C()) = r^a] < negl()$$

where the probability is taken over (g, a).

B   Example of the LSH-PSI protocol

A concrete example of Steps 1.b - 3 of the LSH-PSI PPRL protocol (Figure 6) is given in Figure 7. Suppose that $v = H(455)^{sk_s sk_r}$ then P_s learns via the PSI process that P_r also has a band signature with the same value 455. P_r took care to preserve the order of P_s's encrypted band signatures during the PSI, so P_s can map the shared value v back to the band signature for Band 1 of record N_s, and deduce that P_r has some unknown record that is similar to her own record N_s.
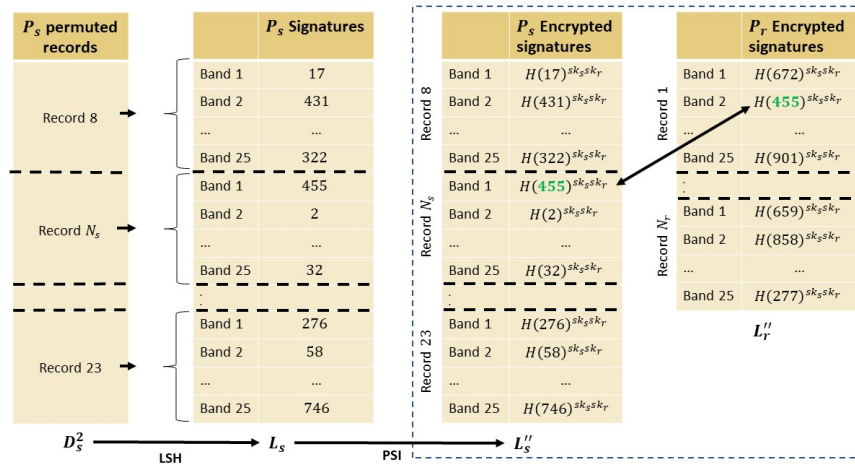

Fig. 7: Steps 1.b - 3 of our protocol. P_s learns via the PSI protocol that the signature for Record N_s Band 1 is shared with P_r.

C   Using the Jaccard indicator

Theorem 1 shows that the LSH-PSI PPRL protocol follows Definition 3 when considering the LSH as the similarity indicator. This means that security reviewers need to accept the privacy leakage that occurs when using an LSH, something that is already done by many organizations that perform RL. However, some reviewers may instead prefer to trust the Jaccard index due to its wide acceptance.

Figure 4 shows two ways to define LSH false-positive events: in relation to exact matches of entire records as in the LSH-PSI PPRL, or in relation to the method of matching pairs of records with a high enough Jaccard index. Thus according to the latter definition an LSH false-positive happens only when a pair of records are matched due to having at least one shared LSH band, and yet they do not have a high enough Jaccard index to justify a claim of similarity. Bounding the false-positive events rate τ' based on the latter definition will allow us to define an LSH-PSI PPRL related to the Jaccard index metric but with a different bound $\tau \cdot \tau'$, where τ is the Jaccard original false-positive bound. In this section, we further discuss the relation between the LSH and the Jaccard index.

For two records s, r with Jaccard index J, Figure 8 shows the probability for an `LSHMatch` $= 1$ event according to Equation 2 with $R = 200$ and $B = 20$. In standard ER solutions, it is the role of the domain expert to decide the specific Jaccard index that would indicate enough similarity between the two records. For example, in the figure the targeted Jaccard index is 0.78. The figure shows the cumulative probability of getting true-positives ($J(s, r) > 0.78$ and `LSHMatch`$(s, r) = 1$), true-negatives ($J(s, r) \leq 0.78$ and `LSHMatch`$(s, r) = 0$), and the corresponding false-positive and false-negative cumulative probabilities.

The above example shows that when $B = 20$ and $R = 200$, it is possible to close the gap between the Jaccard index and the LSH by choosing the Jaccard threshold to be below 0.5. In that case, the probability for a false-positive event is less than 0.0001, which means that one in every ten-thousand records leaks. However, using such a Jaccard threshold will yield many false-positive cases relative to exact record matching, which is less desirable in terms of privacy.

It turns out that it is possible to tune the slope of the accumulated probability function. Figure 9 compares the probability functions in four different setups $B = 20, R = 200$ (setup 1) $B = 100, R = 100$ (setup 2) $B = 14, R = 30$ (setup 3) and $B = 120, R = 18$ (setup 4). Here, we see that replacing setup 1 with setup 2 allows us to set the Jaccard threshold at 0.78 while reducing the LSH false-positive rate to as low as 10^{-8}. However, setup 2 dramatically increases the LSH false-negative rate. Note however that false negatives affect the security less than false-positives, and in addition, users are often much more reluctant to report false positives than to miss reports due to false negatives. Setup 2 may also increase the overall performance of the protocol relative to setup 1 because there are many more bands to encrypt and communicate, as described in the following section.
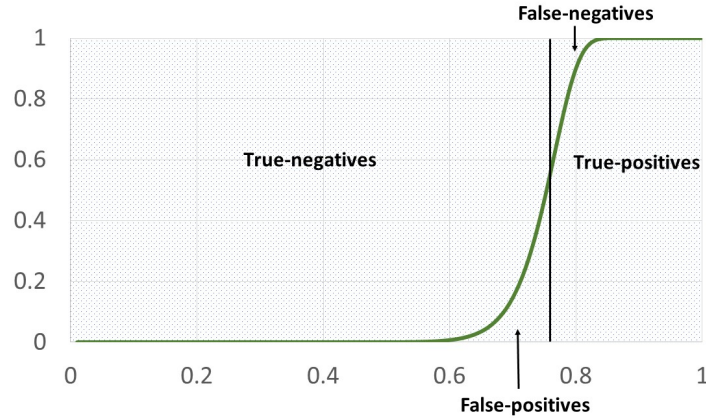
Fig. 8: The function $F(J) = 1 - (1 - J^R)^B$ from Eq. 2, where $R = 20$ and $B = 200$. The black vertical line is the Jaccard index threshold.

C.1 Optimizing the protocol

Setup 4 in Figure 9 probably results in more false-positive and false-negative cases than setup 1, and the low slope of the curve implies a larger region of uncertainty. However, the PSI for setup 4 runs more than 6 times faster than the PSI for setup 1, because there are just 20 rather than 180 band signatures that need to be encrypted and communicated. The change in the R parameter does not affect the performance as much, since it merely determines the number of Min-Hashes that need to be computed locally. It turns out that computing a Min-Hash (like the highly optimized SHA-256 operation) is much faster than computing the power in the the underlying groups of the DH protocol. Moreover, there are known methods for quickly producing R different permutations out of a single SHA-256 call such as the Mersenne twister [3]. Finally, the value of R does not affect the size of the communication.

We use the B and R parameters to control the curve, which in turn affects the protocol's accuracy and performance. Reducing B makes it less likely to find a matching band signature, thus increasing the false-negative probability, but improving performance. The rate of false-negatives can be reduced by decreasing R, thus making it more probable for two bands to match. Conversely, if the false-positive rate is too high, then one can increase R with little performance penalty. We therefore optimize the process by searching for values of B and R that have the minimal B value (for best performance) while more or less preserving the targeted curve shape.

Suppose for example that setup 1 has the targeted probability function. The figure shows the probability function for setup 3, which runs almost twice as fast as setup 1 and has an almost identical probability function. Setup 4 has an
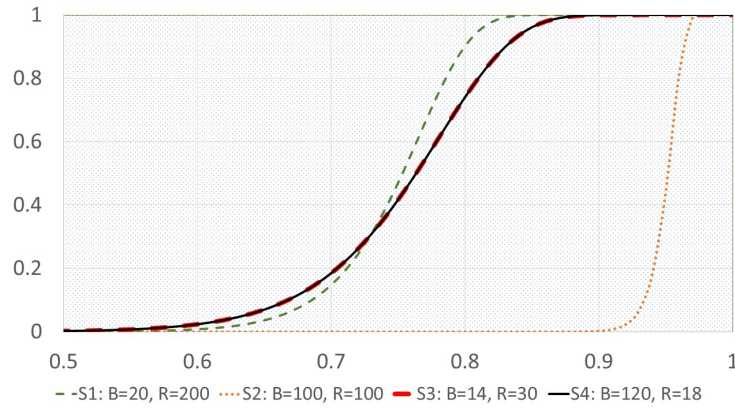
Fig. 9: A comparison of four probability functions $F(J) = 1 - (1 - J^R)^B$ (see Eq. 2) with different B and R values.

almost identical curve as setup 3 so it gives an almost identical accuracy, but it runs much slower because it requires almost 8 times more bands.

C.2   Scoring the reported matches

When a PPRL protocol relies on the Jaccard index but its implementation uses LSH, it may be in the users' interest to quantify the number of false-positive events. To this end, we present a way to estimate the Jaccard index based on the LSH results.

Estimating the Jaccard index for matching pairs When using LSH with B band signatures, it is possible to estimate the actual Jaccard index J by using a binomial confidence interval. By observation 1, the probability for a matching band (i.e. the probability for a match in all R Min-Hashes of the band) is $p = J^R$. Suppose that P_s learns that there are h matching band signatures and $t = B - h$ non-matching band signatures. Using a 95% confidence interval, the Jaccard index lies in the range

$$\left[\sqrt[R]{\left| \frac{h}{B} - 1.96\sqrt{t\frac{h}{B^3}} \right|}, \ \sqrt[R]{\left| \frac{h}{B} + 1.96\sqrt{t\frac{h}{B^3}} \right|} \right] \tag{3}$$

In some cases this interval is too wide, and the users may prefer using a different approach, such as a revealing PPRL. In a revealing PPRL, the two parties learn the intersection of their datasets as in a standard PPRL but they also learn the records of the other parties that are involved in of the intersection. Thus, the leaked information in a revealing PPRL is higher than in a PPRL.

Below, we propose an approach with privacy leakage that lies between the leakage of a revealing PPRL and a PPRL, where we compute the Jaccard index only for matching pairs, without revealing the exact shingles.

Computing the precise Jaccard index for matching pairs Suppose that at the end of the LSH-PSI PPRL protocol, P_s learns the matching pair $(s, Enc(r))$. P_s can ask P_r to participate in another PSI process over the set of shingles of (s, r), where P_s knows s and P_r knows r. In this PSI, P_s only learns the intersection size of the associated shingles $|S \cap R|$ and the size $|R|$, so it can compute $J(s, r) = \frac{|S \cap R|}{|S| + |R| - |S \cap R|}$. Note that learning only the intersection size and not the intersection itself makes it harder for P_s to guess P_r's record.

These additional PSIs are relatively expensive in terms of performance, but we only need to carry them out for the reported matches, which are presumably only a very small fraction of all possible pairs of records. P_s and P_r can decide to perform such PSIs for every matching pair or for selected pairs of special interest, or for pairs selected after estimating the Jaccard index as described above. As mentioned in Section 3 performing a selective PSIs leaks the size of the selection to an eavesdropper **and this should be taken into account in the application threats model**.

D   Our implementation

For reproducibility, we provide concrete details about our LSH implementation. We start by explaining the concept of relative weighting of the record fields.

D.1   Relative weighting of the record fields

Some record fields may be more indicative of identity than other fields. For example, an SSN field is very indicative (though it may also include typos), and a similarity of the full names is more indicative of identity than the similarity of zip codes. A simple method of weighting the effect of the different fields on the matching process is to duplicate the shingles originating from a field for a predefined number of times. We call this number the field weight. For example, consider a PPRL that operates over records with two fields: name and zip code. We use k=6 and k=7 shingles for these fields and set their weights to be 3 and 1, respectively. Then, the 6-shingle 'John S' extracted from the name field 'John Smith' will be duplicated into three separate shingles 'John S1', 'John S2', 'John S3', whereas the zip-code 7-shingle '2304170' will not be duplicated. This causes shingles originating from the name to be three times more likely than zip-code shingles to be the minimum value used by the Min-Hashes of the LSH (see Section 2.2). This will make the band signatures more likely to match if name shingles are identical than if zip-code shingles are identical.

The problem with this shingle duplication weighting method is that the extra shingles slow down the PPRL process because more shingles need to be hashed

by the many Min-Hashes. To this end, we present a novel method for weighting the shingles, which yields the same results as the shingle duplication method but is much faster. The idea is to reduce the hash value of a shingle according to the shingle's weight, to directly increase its chance of being the shingle that receives the minimal value by the Min-Hashes.

We view the hash code h of a shingle as a discrete random variable with uniform distribution over some integer range $[0, maxVal]$. Thus, $x = h/maxVal$ is approximately a random variable with a continuous uniform distribution over $[0, 1]$. Our method relies on this being a good approximation.

Our method is as follows: instead of duplicating a shingle w times, we compute the shingle's hash-code h, normalize it $x = h/maxVal$, then apply the transformation $y = 1 - (1 - x)^{1/w}$, and finally return back to the original scale $h' = \lfloor y * maxVal \rfloor$. Lemma 1 shows that this results with a variable h' whose distribution is the same as the minimum of w independent hashes.

Lemma 1. *Let H_1, H_2, $\ldots$, H_n be i.i.d. random variables with uniform distribution over $[0, 1]$. Let $Y = min(H_1, H_2, \ldots, H_w)$. Then $X = 1 - (1 - H_1)^{1/w}$ has the same distribution as Y.*

Proof. Let F_H be the cumulative distribution function (CDF) of each H_i, i.e., $F_H(h) = h$ in the range $[0, 1]$. Let F_Y be the CDF of Y, i.e., $F_Y(y) = 1 - (1 - F_H(y))^w = 1 - (1 - y)^w$ and its inverse is $F_Y^{-1}(p) = 1 - (1 - p)^{1/w}$, so $X = F_Y^{-1}(H_1)$. The CDF of X is therefore

$$F_X(x) = P(X \leq x) = P(F_Y^{-1}(H_1) \leq x) = P(H_1 \leq F_Y(x)).$$

Since H_1 is a uniform variable over $[0, 1]$, this means $F_X(x) = F_Y(x)$.   □

We observed a 9% speedup when comparing the computation time (ignoring communications) of our PPRL solution using the shingle duplication method versus the above hash-dropping method.

Remark 4. The work in [22] also describes a method of computing a 'Weighted MinHash' over multisets with duplicated elements, but the universe of all possible items (or dimension for vectors) is assumed to be known in advance.

E   LSH description

We are now ready to describe our LSH implementation. The algorithms below use a data structure that we call the field-group data structure FG, which is a list of tuples (s, k, w), where s is a string, $k \in \mathbb{N}$ is the shingles length, and $w \in \mathbb{N}$ is a vector with the shingles' weights, respectively. Algorithm 1 computes the LSH for a given record *record*. First, it concatenates together strings from fields that belong to the same group according to the configuration variable *conf* (Lines 5-6). Then, it attaches to every concatenated string the k, w values of its group as defined by *conf* (Line 7). The algorithm returns the output of the LshFG function on the generated field-group data structure FG (Line 8).

The `LshFG` algorithm uses the auxiliary functions `getWeigthedShingles`, which we describe in Algorithm 2. Its input is a field-group data structure and its output is a list of pairs of k-shingles and their respective weights.

Algorithm 1 Compute the LSH for a given DB record

Input: *record*, a map of fields to values (strings) and *conf* a list of tuples (F, k, w) where F is a set of field names, and $k, w \in \mathbb{N}$ are the shingles length and the fields weight, respectively.
Output: $lsh = [b_1, b_2, \ldots, b_B]$.
1: **procedure** LSH(*record*, *conf*)
2:      $FG = \emptyset$
3:      **for** $t \in conf$ **do**
4:          $s = $ ""
5:          **for** $f \in t.F$ **do**
6:              $s = s \mid record[f]$
7:          $FG = FG \cup (s, t.k, t.w)$
8:      **return** $LshFG(FG)$

Algorithm 2 Returns weighted shingles for given strings

Input: FG a field-group data structure.
Output: *res* an ordered list of pairs (sh, w) where sh is a string and $w \in \mathbb{N}$.
1: **procedure** GETWEIGTHEDSHINGLES(FG)
2:      $res = \emptyset$
3:      **for** $(s, k, w) \in FG$ **do**
4:          $S = $ `getShingles`(s, k) ▷ Returns an ordered list of the k-shingles of s.
5:          $res = res.append\left([(sh, w) \mid sh \in S]\right)$
6:      **return** res

Algorithm 3 describes the function `LshFG`, which basically follows the LSH definition. First, the strings are converted to shingles by invoking Algorithm 2. The loop of lines 8-14 generates the signature bands in M. It starts by computing a 32-bit hash for every shingle (lines 10-11), and then uses them to construct R different hashes for each of the shingles. The R hash values are then reduced according to the shingle weight using the function `CalcH`. This function is based on Lemma 1, where the equation in Line 3 can be modified when $w \leq 2$ to avoid the division and save computations. The resulting R minimal hash values are kept in the M array. To generate fast hash values, we replaced the intermediate `SHA256` calls with a Mersenne twister, which uses random numbers. The algorithm generates and holds these numbers in the arrays C and D. Finally, using `SHA256`, we concatenate and hash the values of M to create the band signature (Line 14).

Algorithm 3 Compute the LSH for a given record field group

  Constants: $MP = 2^{61} - 1$, a Mersenne prime, and $maxVal = 2^{32}$
  Input: $h, c, d, w \in \mathbb{N}$.
  Output: an integer.
1: **procedure** CALCH(h, c, d, w)
2:   $h = \left[h \cdot c + d \pmod{MP} \right] \pmod{maxVal}$
3:   **return** $maxVal \cdot \left(1 - (1 - \frac{h}{maxVal})^{\frac{1}{w}} \right)$ $\triangleright$ based on Lemma 1.

  Input: $B, R \in \mathbb{N}$, and FG a field-group data structure.
  Output: $L = [b_1, b_2, \ldots, b_B]$.
4: **procedure** LSHFG(B, R, FG)
5:   $C \xleftarrow{\$} \{1, \ldots, MP\}^R$
6:   $D \xleftarrow{\$} \{0, \ldots, MP\}^R$
7:   $wS = $ getWeightedShingles(FG)
8:   **for** $b = 1, \ldots, B$ **do**
9:     $i = 1$
10:     **for** $(sh, w) \in wS$ **do**
11:       $H[i + +] = (\text{Trunc}_{32}(\text{SHA256}(sh)), w)$
12:     **for** $r = 1, \ldots, R$ **do**
13:       $M[r] = \min_i \{ \text{CALCH}(H[i].sh, C[r], D[r], H[i].w) \}$
14:     $L[b] = \text{SHA256}(M)$
15:   **return** L
