Secure Blocking for Record Linkage

Kevin Witlox[1]
k.h.d.witlox@student.utwente.nl

Supervisors
Dr. Ing. Florian Hahn[1]
Prof. Dr. Ir. Thijs Veugen[2,1]
Dr. Ir. M.M.J. Stevens[3]
Dr. N. Nicola Strisciuglio[1]

[1]University of Twente, Enschede – Overijssel, The Netherlands
[2]TNO, The Hague – Zuid Holland, The Netherlands
[3]CWI, Amsterdam – Noord Holland, The Netherlands

Abstract – Record linkage is the problem of joining together datasets without a unique identifier. Record linkage can be used to combine multiple data sources for answering research and policy questions, and could therefore be a valuable tool. However, even if the desired answer is only an aggregate, record linkage may be infeasable due to privacy concerns as it requires entire datasets to be shared between parties. This problem may be solved by implementing record linkage as a secure computation, preserving the privacy of the underlying data. Currently however, no blocking technique (a pre-processing step to speed up record linkage) is designed to work as a secure computation, limiting the scalability of secure record linkage solutions. Therefore, we design and implement the first *secure* blocking solution. We compare the running time of our solution against a secure record linkage solution, and show that our secure blocking solution allows for a great reduction in running time.

Keywords – fuzzy matching, approximate matching, privacy-preserving record linkage, privacy-preserving blocking

I INTRODUCTION

In our modern data-driven world many different organizations and institutions process data on natural persons. Governmental institutions keep records of their citizens, banks track the transactions of their customers, and hospitals record the medical information of their patients. These datasets are kept for administrative purposes or to execute the primary task of said institution. However, one can imagine there are many use-cases which could benefit from data analysis on the joint dataset of multiple institutions.

Consider for example medical research. Individual medical institutions may not possess sufficient case data to draw statistically significant conclusions, especially on rare diseases and conditions. However, by combining data from multiple medical institutions one could allow medical researchers to produce new findings [32, 42].

...	...
John Doe	1989-08-07
John Smith	1990-01-01

Dataset A

John J. James	1967-04-12
John Smuth	1990-01-01
...	...

Dataset B

Fig. 1. Simple Example of Record Linkage. The third record of dataset A and the second record of dataset B likely represent the same entity.

Joining datasets in the absence of unique identifiers is a non-trivial task, and is studied as a problem called record linkage. First described in [5] and later formalized in [9], record linkage is the problem of linking data entries across different datasets that refer to the same real world entity (also referred to as Entity Resolution or Data Matching [11]). Figure 1 illustrates this concept. In the figure, there are two datasets, each containing a different representation of the same real-world entity 'John Smith'. The goal for record linkage here is to match these records as the same entity.

Standard record linkage is however unpractical for many use-cases. Sharing vasts amounts of data between organizations to fulfill specific tasks might be undesirable given the possibility of the data leaking or it being used for purposes other than the intended one. Consequentially, many jurisdictions pose limitations on the sharing of data between individuals and organizations, exemplified by Europe's General Data Protection Regulation (GDPR) [38].

To address these issues surrounding data privacy, researchers have investigated a variation on record linkage called privacy-preserving record linkage (PPRL). Here, the goal is to perform the record linkage while keeping Personally Identifiable Information (PII) — *direct identifiers* such as Social Security Numbers, email-addresses but also *quasi-identifiers* such as Birthday or ZIP code — private during this process.

Though many different techniques exists for privacy-preserving record linkage, these techniques often leak at least some amount of information. For example, one technique that has seen real-world deployment [36,

18] is the probabilistic approach using Bloom Filters [41]. This technique relies on the Bloom Filter for privacy, on which cryptanalysis attacks have been shown [25].

Instead, ideally we wish to perform privacy-preserving record linkage without any party learning any information other than its own and the output of the record linkage. This is the promise of Secure Multi-Party Computation (MPC) [47]. Secure Multi-Party Computation is a technique for constructing computations which do not leak any information (under computational assumptions). The technique allows multiple parties, each with private inputs, to jointly compute a function revealing only the output of said function and nothing else. Due to innovations in the field and advancements in computing power, the technique has become more practically viable in recent years.

Recently, Stammler et al. [42] implemented the first full record linkage solution as a secure computation. Their results show good blocking quality performance, but their method lacks a blocking step. Blocking is a pre-processing step for record linkage which partitions the datasets into blocks in such a way that only records within the same block need to be compared, reducing the amount of pair-wise computations needed during the *matching step* [31, 11]. Such a blocking step is needed to speed up the process of record linkage, as the matching step of record linkage is usually expensive.

Blocking is essentially the problem of finding a subset of record pairs from the space of all record pairs $|A| \times |B|$ between two datasets A and B, such that the subset still includes all or most of the matching pairs. This is seemingly incompatible with the standard definition of MPC. The execution of a MPC computation must be indistinguishable between different inputs of the same size, i.e. the memory access patterns and execution paths of a secure computation may not depend on private inputs. Computing on a subset of the private inputs based on other private inputs does not fit in this model.

Recently however, a variation on MPC has gained traction, called Random Access Machine-based Secure Computation (RAM-SC) [33, 13]. In this model, one can construct secure computations with input-dependent memory accesses that run in sub-linear time. In essence, the memory accesses are ran outside of the secure computation, but in such a way that the memory access pattern is indistinguishable from random access pattern of equal length and the memory contents remain secret.

To the best of our knowledge, no MPC blocking method exists as of yet. In this paper, we therefore design and implement the first *secure* blocking solution. With this work, we aim to advance the practicality of using record linkage in the context of secure computation, which would make possible a wide range of possible use-cases.

Contributions Our contributions are as follows:

- We design the first secure blocking solution, and show the adaptations that are needed for blocking to be implemented as a secure computation.
- We implement a LSH-based blocking technique using RAM-SC and manage to block two datasets of 10.000 records in under ten hours.
- We implemented the Square-Root ORAM construction in the MP-SPDZ Framework, a contribution that has been merged into its public git repository.

Outline In the next section we cover the related works in privacy-preserving blocking and secure record linkage, identifying that no secure blocking solution exists yet. We also survey existing blocking techniques for use in our solution. In Section III we introduce the relevant background knowledge on MPC and RAM-SC. In Section IV we introduce the definitions relevant to blocking and formalize the problem. In Section V we design and implement the secure blocking functionality, independently of a particular blocking technique. Then, in Section VI, we describe LSH-based blocking and employ it in our solution. Section VII describes the experimental setup for benchmarking our solution and the results from these experiments. In Section VIII, these results are discussed. Finally, Section IX concludes this paper.

II RELATED WORK

A *Privacy-Preserving Blocking*

There is a large body of research in both privacy-preserving record linkage and privacy-preserving blocking (also called private blocking). Many techniques in this body however either use a weak definition of privacy, or none at all. In this section we briefly cover the literature on privacy-preserving blocking and privacy-preserving record linkage techniques which include a blocking or filtering step.

Al-Lawati, Lee, and McDaniel [26] present one of the first privacy-preserving record linkage solutions which include a privacy-preserving blocking step. Their blocking is an adaptation of Token Blocking. For every unique token of a record, that record is added to the block identified with that token. To preserve privacy, a hashed version of the token is used as the block identifier instead of the plaintext value. A third party Charlie then compares the hashed block identifiers to recognize blocks that both Alice and Bob have in common. The privacy here however must be guarantees through the third party.

A suite of other early protocols for PPRL are those based on the *embedding* approach [39, 46], in which records are embedded into a shared reference space as a vector that preserves the similarity relation between records. These methods reduce the number of pair-wise distance calculations needed between embedded records in a pre-processing step to speed up record linkage. Scannapieco et al. [39] employ a heuristic to rule out certain pairs; Yakout, Atallah, and Elmagarmid [46] map their embedding vectors to a single complex number to serve in a cheap distance calculation. It is assumed that the vector embeddings hide the underlying records. However any leakage in such methods is not well defined [37].

Later, works such as [15, 19] introduce privacy-preserving blocking protocols which employ

conventional privacy definitions to show bounds on the information leaked by their protocols. Inan et al. [15] introduce a hybrid approach for privacy-preserving record linkage, combining k-anonymity based blocking with secure computation for the matching step. First, the datasets are anonymized with k-anonymity, relying on value generalization hierarchies. An initial matching is performed where records with mismatching generalization sequences are filtered out. The remaining pairs are then matched using secure computation.

Another suite of techniques relying on k-anonymity are the approaches based on *reference tables*. Karakasidis and Verykios [19] present a three-party PPRL solution, in which Alice and Bob cluster a reference set using the k-Nearest Neighbour algorithm. The reference set is shared between Alice and Bob, such that Alice and Bob generate the same clustering, but secret to Charlie. Alice and Bob then insert their records into the clustered reference set, sending the obfuscated records together with their cluster identifier to Charlie. Since the clusters are of size k, the information given by a cluster identifier may reflect on at least k records (k-anonymity). Charlie then creates candidate pairs based on the cluster identifiers and continues the record linkage process. Vatsalan and Christen [43] introduce a similar technique based on the 'Sorted Neighbourhood' blocking approach, which outperforms the former. The major drawback of this specific suit of techniques is that they only achieve good accuracy if the reference set is a superset of the datasets [21]. Furthermore, the privacy definition of k-anonymity is known to be vulnerable to attacks [28], especially when attackers have background knowledge — a reasonable assumption when working with dataset on natural persons.

A more strict definition of privacy is that of Differential Privacy [7], which has seen use in more recent works on PPRL [16, 14, 37]. Inan et al. [16] revisit their scheme of [15] and employ this definition. Alice and Bob locally cluster their datasets, and then share the boundaries of these clusters preserving differential privacy by adding noise to these boundaries. This allows Alice and Bob to learn overlapping clusters. Rao et al. [37] extend this idea and prove their solution to uphold the end-to-end differential privacy notion introduced by He et al. [14] by including a third party. Though these techniques are more promising in terms of data privacy than previous works, the statistical leakage might still be prohibitive for certain use-cases.

A recent suite of techniques that has received attention in the context of privacy-preserving blocking due to its speed is the suite of Locality Sensitive Hashing based techniques. Locality Sensitive Hashing techniques work by producing a hash for each record, which preserves some distance function. Thus, more similar records are more likely to be hashed to the same hash. Durham [6] was the first to suggest the use of LSH as a blocking technique for private blocking in conjunction with bloom filters. By sampling random bits from a bloom filter, one creates a H-LSH family (the family of hash functions sensitive to the Hamming distance). Karapiperis and Verykios [22] present a general framework for LSH-based blocking and conclude that H-LSH outperforms Jaccard LSH in terms of speed and precision. Karapiperis and Verykios [21] improve the PQ and RR of this method by counting how often a pair of records produces hash collisions. Ranbaduge et al. [35] utilize LSH differently. The propose a multi-party protocol where each party locally clusters their records, which are encoded as bloom filters. Each party then generates a signature for each local cluster using MinHash, producing cluster representatives. To avoid pair-wise comparisons between cluster representatives of all parties, LSH is used to map similar cluster representatives to the same block. The disadvantage of these techniques is however that the leakage of the hashes is not defined.

B *Secure Record Linkage and Blocking*

With secure record linkage and secure blocking, as opposed to privacy-preserving record linkage and privacy-preserving blocking, we refer to techniques which employ Secure Multi-Party Computation.

Though sparingly, MPC has been used in the literature of PPRL. Most methods employing MPC however, do not use MPC for the entire process such that the entire process in provably secure. In Inan et al. [15] for example, the authors use MPC to implement a matching step for record linkage, but use a k-anonymity based method for blocking. Lazrig et al. [27] implement their matching step using MPC, which calculates securely the distance between Bloom Filters, but information is revealed selectively to allow for blocking. Essex [8] introduces a novel cryptographic primitive to perform Bloom Filter distance calculations securely, but their method does not constitute a full record linkage solution nor takes into account blocking.

Most recently, Stammler et al. [42] introduce the first full secure record linkage solution, taking into account the varying types and weights of attributes. They implement their solution using the ABY-framework and achieve a high record linkage quality. Their solution however does not include blocking.

C *Blocking*

In this work, we investigate the feasibility of implementing blocking as a secure computation. We will be relying on the vast body of existing blocking methods to find a method suitable in this setting. In this section we briefly cover the main techniques in the blocking literature. We describe the general or the simplest implementation, but forgo mentioning all methods that exist within the techniques. The categorization is based on the surveys done by Durham [6], Schnell [40], and O'Hare, Jurek-Loughrey, and Campos [31].

This subsection uses the common blocking metrics Reduction Ratio (RR), Pairs Quality (PQ) and Pairs Completeness (PC) to describe the blocking quality of the mentioned techniques. These metrics correspond to the relative reduction in needed comparisons, the precision of the generated pairs and the recall of the generated pairs

respectively. See Section VII for the complete definitions.

Standard Blocking The first blocking method is called *Standard Blocking*, introduced together with the notion of record linkage itself by Fellegi and Sunter [9]. In Standard Blocking, each record of a dataset is mapped to one or multiple *blocking keys* using some blocking mechanism. A blocking key is created via a rule which composes one or multiple attributes or parts thereof. A simple example is to take the attribute 'year of birth' as the blocking key.

Standard blocking has two main drawbacks. First, its performance greatly depends on the selection of the blocking keys, which have to be chosen based on expert knowledge. Second, an error on the blocking key will result in missed matches. To counteract this, the linkage is repeated for multiple different blocking keys.

Token Blocking *Token Blocking* is another older technique. The technique is schema-agnostic and considers a record as a plain bag-of-words, i.e. a collection of all words of all attributes in the record. Each unique word is a token that identifies a block. Thus, a single record is mapped to potentially many blocks, one for each unique word in the record. The method achieves a high Pairs Completeness, as matching records commonly have at least one word in common. However, due to the many tokens that can exist in a dataset, the method has a low Reduction Ratio. Due to this fact, we do not consider it for our secure blocking scheme.

Clustering In 2000, McCallum, Nigam, and Ungar [29] introduced a clustering technique called *Canopy Clustering*, which has subsequently been used in the blocking literature. Say we have a large dataset, a computationally cheap distance measure for records within this dataset and two thresholds t_1 and t_2. The idea is to create clusters (canopies) at random containing similar records according to the distance measure, such that all records are contained in at least one canopy. The clustering is performed by picking at random a record (the centroid) from a list, initially holding all records, and finding all records within some loose distance threshold t_1. Then, all records in the newly formed canopy within a tight distance threshold t_2 are removed from the list. This processes is repeated until the list is empty.

One disadvantage of *Canopy Clustering* is that its performance relies greatly on the choice of parameters for t_1, t_2, as well as the similarity function, and the resulting number of canopies. Depending on the parameter choice, one may create few large clusters resulting in better Pairs Completeness but worse Reduction Ratio, or many smaller clusters resulting in higher Reduction Ratio but worse Pairs Completeness. Secondly, even though a cheap distance measure is used, pair-wise distance computations need to be performed for each centroid with all other records, which may be infeasible for large datasets.

Sorted Neighborhood Various *sorting*-based methods have been proposed for blocking, the most common of which is the *Sorted Neighborhood* approach [6]. In its simplest form, one derives a blocking key and sorts the entire dataset with this blocking key. Then, the dataset is processed under a sliding window, where all records within the window are considered to be in the same block. Various adaptions have been developed. Some methods leverage multiple passes with different blocking keys to increase the Pairs Completeness when errors may occur in the blocking key. Other techniques consider an adaptive window size set according to the similarity between the first and last record in the window, to optimize both the Pairs Completeness and the Reduction Ratio.

Locality Sensitive Hashing Locality Sensitive Hashing was initially introduced by Indyk and Motwani [17] to solve the *approximate nearest neighbor problem*. It works by hashing items using hash functions with the special property that the probability of hash collisions, i.e. two items producing the same hash, is higher for items that are similar. This technique has successfully been used for (privacy-preserving) *blocking* [6, 22, 21].

III PRELIMINARIES

In this section we will introduce the relevant background knowledge on the building blocks needed for this paper. Section III.A gives an introduction on the standard definition of Secure Multi-Party Computation (MPC). Section III.B introduces RAM-based Secure Computation (RAM-SC), the variant of MPC used in this paper, and Section III.C introduces the Oblivious RAM (ORAM) — the main building block for RAM-SC. Section III.D introduces Bloom Filters, which are used in Section VI as part of the blocking solution. Finally, Section III.E introduces the MP-SPDZ Framework, which is used to implemented the design proposed in this paper.

A *Secure Multi-Party Computation*

Secure Multi-Party Computation (MPC) offers a method for computing a function securely. It works by replacing a trusted-third party with a cryptographic protocol. This allows multiple parties holding sensitive data to compute the function over the combined set of data without revealing any information other than the output of the function to the participating parties. More formally, given n parties each owning a secret x_i, Secure Multi-Party Computation allows the computation of function $f(x_1, \ldots, x_n)$ without revealing any of the private inputs x_i.

There are a variety of Secure Multi-Party Computation protocols, for both boolean and arithmetic inputs. However, in general these protocols apply the same principle. In this paper we limit ourselves to record linkage between two parties, and thus we set $n = 2$, resulting in parties P_1 and P_2. The basic principle is that a secret input v of some particular party is shared between all n parties, such that the parties can jointly compute on this input without directly learning the value of the input. To illustrate, P_1 would like to secretly share private input v with P_2. P_1 samples a random value r which it uses

to create two shares of v by setting $\langle v_1 \rangle = v - r$ and $\langle v_2 \rangle = r$, the latter it sends to P_2. It is easy to see that $\langle v_1 \rangle + \langle v_2 \rangle = v$ and neither $\langle v_1 \rangle$ or $\langle v_2 \rangle$ alone gives any information on v due to the random mask r. We say that P_1 has *secret shared* v with P_2, where P_1 holds $\langle v \rangle_1$ and P_2 holds $\langle v \rangle_2$.

In the remainder of this paper, we will use the following notation. A value v that is *secret shared* is denoted by $\langle v \rangle$, implying that all participating parties $P_i \in \{P_1, \ldots, P_n\}$ own the share $\langle v_i \rangle$ of value v. We will assume any Secure Multi-Party Computation protocol provides two algorithms share and reveal. The former creates shares from a secret value, denoted $\langle v \rangle \leftarrow \mathsf{share}(v)$ and the latter combines the shares to reveal the underlying value, denoted $v \leftarrow \mathsf{reveal}(\langle v \rangle)$.

B *RAM-based Secure Computation*

Advances in both the efficiency and available tooling in Multi-Party Computation are alleviating the costs and burdens that are associated with performing general secure computations, paving the way for practical adoption of the technology. However, one challenging aspect that remains in compiling arbitrary programs into secure computations is that of *input-dependent* memory accesses. The problem lies in the requirement that no information may be leaked about any intermediate value of the computation, and thus the instructions executed during the computation may not vary based on the private inputs.

Now suppose that two parties involved in a secure computation share a secret index i, with which they need to access the i'th element in some secret shared memory. This is analogous to a random access memory (RAM) instruction. Within the well-known circuit-based MPC-implementations [47], the solution is a linear search, requiring the computation to 'touch' every memory element in order to hide the element actually being read or written. This prevents algorithms from achieving sub-linear running time complexity in the size of the inputs, making computations involving large inputs (e.g. databases) infeasible.

Ostrovsky and Shoup [33] observed that a secure computation requiring random access memory could be realised efficiently using Oblivious RAM. Gordon et al. [13] developed this idea and proposed the first general framework for so-called RAM-based Secure Computation (RAM-SC). In short, the ORAM randomizes the access pattern to a RAM, such that an observer cannot distinguish access patterns of the same length for different inputs. In RAM-SC, the ORAM construction is secret-shared between the parties. To perform a memory access to a secret index, the parties emulate an ORAM access within the secure computation. Then, the physical memory location resulting from the access protocol is revealed, which can be used to read or write the value at this memory location from within the secure computation. Since an ORAM access generally has a time complexity sub-linear in the size of the memory, this construction can be used to build secure computations that scale in sub-linear time in the size of the private inputs.

Though the asymptotic performance of RAM-SC for memory accesses is a great improvement compared to linear scan, the practical performance impact of initializing and emulating an ORAM within a secure computation does hinder its usage. As a result, Wang et al. [45] noted that the current ORAM literature, which focussed on improving the asymptotic complexity memory accesses in terms of bandwidth overhead, had little impact on the practical performance of RAM-SC. The authors noted that in the context of secure computations, one should optimize for a low circuit complexity. A line of work was spawned that investigates ORAM-constructions specifically in the secure computation setting [13, 10, 45, 44, 48, 4].

In this paper, we will make use of the Square-Root ORAM construction introduced by Zahur et al. [48]. The Square-Root ORAM construction benefits from a specialized initialization protocol, whereas previous constructions initialize by inserting the initial dataset one-by-one via the access protocol [44]. Furthermore, though the asymptotic complexity of Square-Root ORAM is worse than the previous construction by Wang, Chan, and Shi [44], its running time is faster for block amounts under 2^{14} — which is relevant for our design.

In the next section we give more details on how an Oblivious RAM construction works and outline the setup of the Square-Root ORAM used in this paper.

C *Oblivious RAM*

Oblivious RAM (ORAM) was first introduced by [12] as a construction aimed at hiding the *access pattern* by a processor to its memory within a computer. The access pattern – the order and location of `read` and `write` instructions to memory – can reveal to an observer the structure of a program running on the processor (e.g. loops) and values used in the computation, if these influence the memory access pattern.

To achieve this, the *access protocol* of an Oblivious RAM translates a *virtual memory instruction*, i.e. the instruction as needed by the processor, into one or multiple *physical memory instructions*, i.e. actual `read` and `write` instructions to the memory. The series of physical instructions hide the virtual instruction being issued.

Notation We use the following notation to describe interaction with a generic Oblivious RAM. An Oblivious RAM is initialized using the initalization function

$$\mathtt{oram} \leftarrow \mathsf{Oram.init}(n, l, D),$$

which takes in as parameters the required size of the memory n, the size of each memory location (length) l and the dataset D to initialize the memory with, returning the state of the Oblivious RAM.

An Oblivious RAM is accessed using the access function

$$v \leftarrow \mathsf{Oram.access}(\mathtt{oram}, i, w, v),$$

which takes as parameters the state of the ORAM upon which to act, the secret virtual location $i \in \{0, \ldots, n\}$ to access, a boolean w indicating whether the instruction should write to the virtual location, and the value v which is written if w is true. The output is the (newly written) value at virtual location i.

1 *Square-Root ORAM*: Zahur et al. [48] revisit the *Square-Root ORAM* construction introduced in [12] and adapt it to the secure computation setting. In this section we briefly describe the original algorithm and the adaptions made in [48].

The idea of Square-Root ORAM is to perform a secure shuffle on an encrypted memory, in order to hide to the server the virtual location and contents the client is accessing. The main complexity of the algorithm lies in not revealing repeated access to the same virtual location.

To achieve this, the client keeps a local stash hidden to the server. Whenever the client accesses a virtual location, the contents are retrieved from the physical location and also stored in the stash. If the client repeats a requests to the same virtual location, the actual content is retrieved from the stash, and an independent physical location is accessed in the physical memory, revealing no information to the server. When the stash is full after exactly T accesses, the ORAM is reinitialized (refreshed). The stash size is equal to this *refresh period* T, which is set to the square-root of the ORAM size.

D *Bloom Filters*

The Bloom Filter, introduced by Bloom [2] in 1970, is a space-efficient probabilistic data-structure for representing a set in order to support membership queries. A membership query returns whether a given element is contained within a dataset. The solution is space-efficient in that the Bloom Filter data-structure maps a set with arbitrarily large elements to a comparitively limited sized bit-array by utilizing hashing.

The Bloom Filter itself is an array of m bits. Initially, all bits are set to 0. To add an element x to the data-structure, one hashes the element with k independent hash functions $\boldsymbol{h} = \{h_1, \ldots, h_k\}$, each hash function producing an index in the range of the bit array $\{1, \ldots, m\}$. Then, for all hash functions $1 \leq i \leq k$ the bit $h_i(x)$ is set to 1. Figure 2 illustrates hashing the elements $\{\_S, SM, MI, IT, TH, H\_\}$ and $\{\_S, SM, MY, YT, TH, HS, S\_\}$ to a Bloom Filter of $m = 12$ bits with $k = 2$ hash functions.

To query whether an element is contained in the Bloom Filter, we hash this element using the same family of hash functions $\boldsymbol{h}$. We then check all bits in the Bloom Filter indexed by the outcomes of the hash functions. If one or more indexed bits are unset, the element cannot be in the set, as hash functions are deterministic. If all bits are set however, the element may be in the set, or another (collection of) element(s) collided on each of those bit

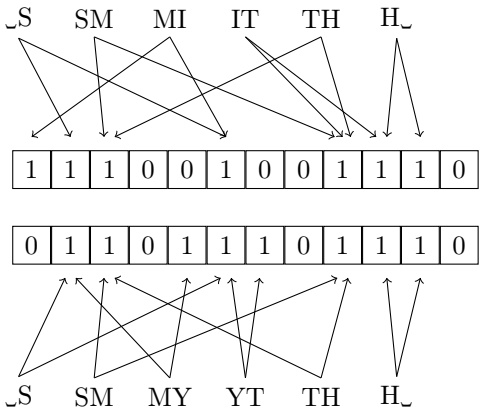

Fig. 2. Visual example of two bloom filters, the top bloom filter encoding 'SMITH' and the bottom bloom filter encoding 'SMYTHS'. The bloom filters are parameterized with $k = 2$ hash functions and $m = 12$ bits. Picture recreated from [42].

positions resulting in a false positive. For a set S of size $|S| = n$ this false positive rate is given by

$$f = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k.$$

E *MP-SPDZ Framework*

To implement the secure computation step of our scheme, we utilize the MP-SPDZ Framework by Keller [23]. The MP-SPDZ framework provides a high-level interface for constructing protocol-independent secure computations, ideal for benchmarking scenarios. Secure computations constructed in the high-level Python interface are compiled into an intermediate assembly-like representation. This assembly is then interpreted by a virtual processor that implements a specific cryptographic protocol to execute the computation securely. The framework includes different cryptographic protocols for different attacker models.

The framework included implementations of TreeORAM, PathORAM and CircuitORAM [24, 44], but not of Square-Root ORAM. We implemented this ORAM construction and merged it into the public git repository of MP-SPDZ.

IV PROBLEM STATEMENT

In this section we introduce the necessary definitions for blocking and formalize the problem. In our definitions for blocking, we limit ourselves to a particular model of blocking which will combine well with secure computation.

A *Blocking*

Blocking is a pre-processing step in the record linkage pipeline (see Figure 4) that aims to reduce the number of pair-wise comparisons between records needed in the matching step. To this end, the blocking method formulates a set of records of its own, called the set of *candidate record pairs* $\tilde{\mathcal{M}}$. The idea is that calculating the set of candidate record pairs is much cheaper than directly calculating the set of matching records $\mathcal{M}$ on the entire dataset.

At a high level, blocking does the following [11]. Assume we have two datasets A and B following the same database schema $\mathcal{I}$. By blocking the dataset, we partition the joint set of records $A \cup B$ into groups called blocks. Only those records $a \in A,\ b \in B$ that find themselves in the same group are added to the set of candidate record pairs $\tilde{\mathcal{M}}$, resulting in fewer comparisons needed in the matching step. An example of a simple blocking strategy is to group all records by their ZIP-code, then adding only those records to $\tilde{\mathcal{M}}$ that have the same ZIP-code.

Definition IV.1 (Schema)**.** *A dataset schema $\mathcal{I}$ is the power set of datasets that have the same attributes. A dataset schema is described by a vector of attributes, denoted as $(I_1, \ldots, I_d)$. A record b in a dataset B of schema $\mathcal{I}$ is a vector of size d, in which element i, $1 < i \leq d$, is addressed as $b.I_i$.*

B *Model of Blocking*

In this paper, we aim to construct a blocking scheme that efficiently runs as a secure computation. Secure computation is orders of magnitude more computationally expensive to compute than non-secure computation. Since blocking is a technique used to scale record linkage to large datasets, we should aim to minimize the computation necessary in the secure domain.

To this end, we reduce our scope of blocking to blocking techniques producing independent blockings (such as [9, 22, 21]). In this model, the result of assigning a single record to a particular block is independent of the datasets involved in the blocking process (as apposed to e.g. clustering or sorting based methods). What remains of the computation needed in the secure domain is determining which records coincide in the same blocks. We describe this model as consisting of one or more *blocking keys* that constitute a *blocking scheme*.

Simply put, a blocking key is a function that maps a record to a block. To illustrate, consider the blocking key '[Third letter of Surname]'. This blocking key has a co-domain of 26 blocks. Applying this blocking key to a record with surname 'Smith' results in the identifier 9, representing the 9th letter 'i' in the alphabet.

Definition IV.2 (Blocking Key)**.** *Let $\mathcal{I}$ be a schema. A* blocking key bk *is a function $\mathsf{bk} : \mathcal{I} \to \mathbb{Z}$ that maps a record onto a* block index *or* block identifier*.*

We introduce some additional notation for applying a blocking key to an entire dataset. Assume we have a dataset B and a blocking key bk. We denote the result of applying bk to every record in B as $\mathcal{B}^{B,\mathsf{bk}}$, the 'blocked dataset'. This blocked dataset is a multiset of blocks, each block containing all records that are mapped to that block by the blocking key bk. For block h, we have that

$$\mathcal{B}_h^{B,\mathsf{bk}} = \{b \in B : \mathsf{bk}(b) = h\}.$$

Now consider what happens in case of a typographical error. Assume the entity 'John Smith' is contained the dataset of both Alice and Bob, but Bob's record is erroneously set to 'John Smyth'. Our blocking key would group these two records into different blocks, resulting in a missed match of these records.

...	...
John Doe	1989-08-07
John Smith	1990-01-01

Dataset A

John James	12/04/1967
Jon Smith	01/01/1990
...	...

Dataset B

Fig. 3. Simple Example of Record Linkage

To improve the completeness of this blocking strategy, we may add a second blocking key to the *blocking scheme*, e.g. *[First letter of first name]*. A blocking scheme, denoted BK, is a collection of blocking keys. If our strategy is to perform record linkage on all records overlapping in at least one block, this would allow us to catch such a typographical error.

With multiple blocking keys, we must also specify the exact condition determining when to consider a pair of records as a candidate record pair. To this end we introduce a threshold t, where

$$0 < t \leq \Gamma,\ \Gamma = |BK|,$$

specifying exactly *how many* blocks should overlap between a record pair (a, b), before the pair is counted as a candidate pair.

For easy of notation, we introduce a function

$$\begin{aligned} &\mathsf{collisions} : A \to \mathbb{N}_\Gamma \\ &(a,b) \mapsto \left|\left\{\mathsf{bk}_\gamma \in BK : b \in \mathcal{B}_h^{B,\mathsf{bk}_\gamma}, h \leftarrow \mathsf{bk}_\gamma(a)\right\}\right|, \end{aligned}$$

which given a pair of records a, b returns the number of blocking keys for which a and b are mapped to the same block index, i.e. the number of collisions between a and b.

C *Problem*

Following the description of the previous section, we then obtain the following definition of blocking:

Problem 1 (Blocking)**.** *Let $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_m\}$ be a datasets according to schema $\mathcal{I}$. Let $BK = \{\mathsf{bk}_1, \ldots, \mathsf{bk}_\Gamma\}$ be a blocking scheme. Find the set of candidate record pairs*

$$\tilde{\mathcal{M}} = \left\{(i,j) : |\mathsf{collisions}(a,b)| \geq t,\ a_i \in A,\ b_j \in B\right\}.$$

that includes every record pair mapped to the same block at least t times.

V FRAMEWORK FOR SECURE BLOCKING

In this section we describe our general solution to implementing blocking securely. We first describe the basic idea of implementing blocking in the context of secure computation in Section V.A. Next, in Section V.B, we formalize the functionality to implement. Section V.C describes what exactly we mean by *secure* blocking, and finally Section V.D details how to implement this functionality. Section V.E and Section V.F analyze and describe respectively the complexity of and optimizations to the implementation.

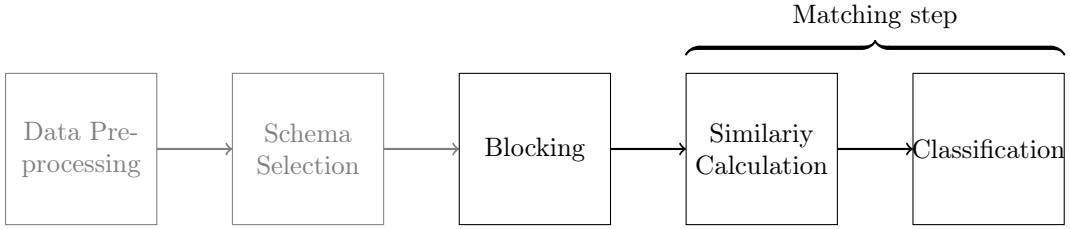

Fig. 4. Overview of a record linkage pipeline. Data Preprocessing and Schema Selection are not considered in this paper.

A *Idea*

First we present the idea behind our solution. For ease of exposition, we fix $|BK| = 1$ and $t = 1$ in this section.

Recall that the blocking keys and the resulting blocks are *independent* of the input data, i.e. Bob does not need to know the contents of A in order to compute the blocking key over his dataset or vice-versa. Note that this is a consequence of the blocking model we described, not an inherent property of all blocking techniques.

Now we take the basic case where Alice only holds a single record a. The output of a blocking solution in this case are the pairs (a, b) for all record of bob $b \in \mathcal{B}_h^{B,\mathsf{bk}}$ that reside in block $h = \mathsf{bk}(a)$.

The obstacle here is that h depends on secret data a. Accessing a secret index h in a collection $\mathcal{B}^{B,\mathsf{bk}}$ cannot be computed efficiently in the standard MPC model. Instead, we use the RAM-SC model, which allows us to efficiently perform this operation using Oblivious RAM.

B *Functionality*

The overview above describes how we efficiently retrieve the record pairs that coincide in the same block. This however, does not solve the entire problem. Problem 1 requires that the result set contains only the candidate record pairs, meaning that we must find a way to filter out the pairs which reach the threshold of t collisions on multiple blocking keys. If we wish to calculate this set exactly, we would have to keep and update a dynamically sized result set, as the amount of pairs that reach the threshold cannot be known in advance. This however is expensive and difficult in the context of secure computation.

To get around this, we slightly alter the output of the functionality. Instead of adding all pairs such that $\mathsf{collisions}(a, b) \geq t$, we add for each record $a \in A$ the top x record pairs $(a, \cdot)$ ranked by the number of collisions of each pair as defined by $\mathsf{collisions}(a, b)$. With a fixed and publicly known x results per record $a \in A$, we can unconditionally add these records to $\tilde{\mathcal{M}}$. The downside is that we introduce another parameter which influences the quality of the blocking process as we will discuss in Section VIII.

This brings us to the following functionality for blocking, which we we implement using the MP-SPDZ Framework.

Functionality 1 (Blocking)**.** *Let $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_m\}$ be a datasets according to schema $\mathcal{I}$. Let $BK = \{\mathsf{bk}_1, \ldots, \mathsf{bk}_\Gamma\}$ be a blocking scheme. Given private inputs A and B, public inputs $n = |A|, m = |B|$ and parameter $x,\ 0 < x \leq m$, find the set $\tilde{\mathcal{M}}_x$*

$$\max(a, B) = \arg\max_{1 \leq j \leq m}(\mathsf{collisions}(a, b_j))$$
$$\mathrm{top}_x(a, B)\ : X_0 = \emptyset$$
$$X_{x+1} = X_x \cup \{\max(a, B \setminus X_x)\} \text{ for } x \geq 0$$
$$\tilde{\mathcal{M}}_x = \{(i, j) : a_i \in A, j \in \mathrm{top}_x(a_i, B), \}$$

where $|\tilde{\mathcal{M}}| = n \times x$, containing for each record $a \in A$ the top x pairs out of all possible pairs $\{(a, b) : b \in B\}$ according to the number of collisions.

C *Threat Model*

We wish to *securely implement* Functionality 1 [1]. In the ideal world, Alice and Bob send A and B to a trusted third party $\mathcal{T}$, who then computes the Functionality 1 and sends back only $\tilde{\mathcal{M}}$. In this ideal world, each party learns $\tilde{\mathcal{M}}$ and nothing else. Using MPC, we replace this imaginary trusted third party $\mathcal{T}$ with a cryptographic protocol.

In this paper we consider two-parties, which limits the relevant cryptographic protocols to those for the 'dishonest-majority' setting, as there can be no 'honest-majority'. Furthermore, we assume that the parties behave in a 'semi-honest' fashion. This means that a party will try to learn as much as it can from the data it obtains during a protocol run, but will not deviate from the protocol. In use-cases for blocking such as research or cooperation of institutes within the same jurisdiction, this should be sufficiently strong. In such a setup, there is trust between the institutions but the problem is keeping the data private. Since our implementation uses the MP-SPDZ Framework, it is trivial to run the computation using a different cryptographic protocol, e.g. a protocol guaranteeing security in the malicious attacker model. However, this will severely impact the running time, as malicious attacker protocols require additional checks during the computations.

The aforementioned security assumptions limit the relevant cryptographic protocols implemented in the MP-SPDZ Framework to the 'semi', 'hemi', 'temi' and 'soho' protocols. In our implementation we use the 'semi2k' protocol (which resembled integer calculation of 64-bit processor for $k = 64$), as this resulted in the best running time performance. This protocol is the result of stripping the SPDZ2k protocol from all elements needed for malicious security, leaving the generation of additively shared Beaver Triples using Oblivious Transfer [23].

D *Full Procedure*

In this section, and the accompanying pseudo-code, the notation $(x_1, \ldots, x_n)$ is used to denote a n-sized tuple, an ordered sequence of objects.

Alice and Bob both own a dataset according to the dataset schema $\mathcal{I}$, datasets A and B respectively. Before starting the secure computation section of the algorithm, the parties must first perform local computation to transform their dataset into a suitable structure, which differs between Alice and Bob.

In lines 2-5, Alice constructs her input from her dataset A such that each record in $a \in A$ has a corresponding tuple in A'. Each tuple contains $1 + \Gamma$ items, one for every blocking key bk_γ and the index of a. The resulting tuple has the form $(i, \mathsf{bk}_1(a_i), \ldots, \mathsf{bk}_\Gamma(a_i))$. The index i corresponds to the position of a_i in dataset A. This index does not need to be secret, as Bob cannot learn anything from the indexes alone other than the size of Alice's dataset. In line 13, the structure A' is secret-shared with Bob.

In lines 6-12, Bob creates the inverse mapping of Alice's structure. For every blocking key bk_γ, Bob stores a mapping from values in its co-domain $h \in \mathsf{bk}_\gamma$ to records $\{b : \mathsf{bk}_\gamma(b) = h\}$ mapped to this value. For efficient computation, this mapping is stored as a tuple of $|B| = m$ bits. In this tuple, the j'th bit indicates whether record b_j is included in block $\mathcal{B}_h^{B,\mathsf{bk}_\gamma}$. The resulting mapping T^γ has a size of $|\mathsf{bk}_\gamma| \times m$, holding a m-sized tuple of bits for every value h in the co-domain of bk_γ. Each mapping T^γ is secret-shared with Alice and then used to initialize a corresponding ORAM $oram^\gamma$ (lines 15 and 16).

Lines 18-28 are the core of the secure computation. For every record $a \in A$, we wish to find the records of Bob that collide at least t times. To this end, we keep track of the collisions using a m-sized tuple of counters initialized on line 20. For every blocking key bk_γ, we retrieve the block $\mathcal{B}_h^{B,\mathsf{bk}_\gamma}$ where $h = \mathsf{bk}_\gamma(a)$, i.e. Bob's records that are blocked to the same block as Alice's record. Since the *block* retrieved is a m-sized tuple of bits, this tuple can be added element-wise to the *counters* tuple.

After all the blocks for record a have been retrieved, the *counters* tuple needs to be interpreted and used to populate the set $\tilde{\mathcal{M}}$. This task is performed by the procedure GetTopMatches, taking in the context variables i, *counters* and $\tilde{\mathcal{M}}$ and the parameter x. As described in Section V.B, we retrieve for record a the top x matching records $b \in B$. The procedure starts by splitting the m-sized tuple of counters in half (resulting in tuples $left$ and $right$), and then performing an element-wise secure comparison between the two resulting tuples. Recall that the j'th element in *counters* represents the counter for record pair (a, b_j). The procedure now writes back the ids of the $\frac{m}{2}$ highest counters, according to the results of the secure comparison.

As we will discuss in Section VIII, the result of GetTopMatches is not guaranteed to be correct but due to distribution of values in *counters* works well as a trade-off between recall and running time.

Algorithm 1 Pseudocode for Secure Blocking

1: **procedure** SecureBlocking(A, B, BK, x)
where $A = \{a_1, \ldots, a_n\}, B = \{b_1, \ldots, b_m\}$,
$BK = \{\mathsf{bk}_1, \ldots, \mathsf{bk}_\Gamma\}$

▷ Local Computation - Alice

2: $A' \leftarrow \emptyset$
3: **for** $a_i \in A$ **do**
4: $A' \leftarrow A' \cup (i, \mathsf{bk}_1(a_i), \ldots, \mathsf{bk}_\Gamma(a_i))$
5: **end for**

▷ Local Computation - Bob

6: **for** $\gamma \leftarrow 1$ **to** Γ **do**
7: $T^\gamma \leftarrow \mathbf{0}^{|\mathsf{bk}_\gamma| \times m}$
8: **for** $b_j \in B$ **do**
9: $h \leftarrow \mathsf{bk}_\gamma(b_j)$
10: $T^\gamma_{hj} \leftarrow 1$
11: **end for**
12: **end for**

▷ Secure Computation

13: $\langle A' \rangle \leftarrow \mathsf{share}((A'))$
14: **for** $\gamma \leftarrow 1$ **to** Γ **do**
15: $\langle T^\gamma \rangle \leftarrow \mathsf{share}(T^\gamma)$
16: $\langle oram^\gamma \rangle \leftarrow \mathsf{Oram.init}(n, l, \langle T^\gamma \rangle)$
17: **end for**

18: $\tilde{\mathcal{M}} \leftarrow \mathbf{0}^{n \times m}$
19: **for** $(i, \langle \mathsf{bk}_1(a_i) \rangle, \ldots, \langle \mathsf{bk}_\Gamma(a_i) \rangle)$ **in** $\langle A' \rangle$ **do**
20: $\langle counters \rangle \leftarrow \emptyset$
21: **for** $\gamma \leftarrow 1$ **to** Γ **do**
22: $\langle h \rangle \leftarrow \langle \mathsf{bk}_\gamma(a_i) \rangle$
23: $\langle block \rangle \leftarrow \mathsf{Oram.access}(\langle oram^\gamma \rangle, \texttt{read}, \langle h \rangle)$
24: $\langle counters \rangle \leftarrow \langle counters \rangle + \langle block \rangle$
25: **end for**

26: $\tilde{\mathcal{M}} \leftarrow \mathsf{GetTopMatches}(i, \langle counters \rangle, \langle \tilde{\mathcal{M}} \rangle)$
27: **end for**
28: **end procedure**

29: **procedure** GetTopMatches$(i, \langle counters \rangle, \langle \tilde{\mathcal{M}} \rangle)$
30: $r \leftarrow |\langle counters \rangle|$
31: $\langle ids \rangle \leftarrow (\langle 1 \rangle, \ldots, \langle m \rangle)$
32: **while** $|\langle counters \rangle| > x$ **do**
33: $mid \leftarrow \lfloor \frac{r}{2} \rfloor$
34: $\langle left \rangle \leftarrow (\langle counters \rangle, \ldots, \langle counters_{mid} \rangle)$
35: $\langle right \rangle \leftarrow (\langle counters_{mid+1} \rangle, \ldots, \langle counters_{mid \cdot 2} \rangle)$
36: $\langle cond \rangle \leftarrow (\langle left \rangle, \ldots, \langle left_{mid} \rangle) \overset{?}{>} (\langle right \rangle, \ldots, \langle right_{mid} \rangle)$
37: $\langle ids \rangle \leftarrow (\langle cond_1 \rangle \cdot (\langle ids_1 \rangle - \langle ids_{mid+1} \rangle) + \langle ids_{mid+1} \rangle, \ldots, \langle cond_{mid} \rangle \cdot (\langle ids_{mid} \rangle - \langle ids_{mid \cdot 2} \rangle) + \langle ids_{mid \cdot 2} \rangle)$
38: $r \leftarrow \lfloor \frac{r}{2} \rfloor$
39: **end while**
40: $\langle \tilde{\mathcal{M}}_{i\cdot} \rangle \leftarrow \langle \tilde{\mathcal{M}}_{i\cdot} \rangle + (\langle id_1 \rangle, \ldots, \langle id_x \rangle)$
41: **end procedure**

TABLE I

The local computation, secure computation, communication and round complexities of Algorithm 1

Local	Secure	Comm.	Rounds
$\mathcal{O}(\Gamma \times (n+m))$	$\mathcal{O}(n \times m)$	$\mathcal{O}(n \times m)$	$\mathcal{O}(n \log m)$

E *Complexity*

In this section we analyze the complexity of the implementation. Recall that the sizes of the inputs are denoted as $n = |A|, m = |B|$, the amount of blocking keys as $\Gamma = |BK|$.

First, the time complexity of the local computation. Both Alice and Bob need to 'block' their records, i.e. apply each blocking key function to each of their records, resulting in $\Gamma \times n$ computations for Alice and $\Gamma \times m$ computations for Bob. The time complexity of the local computation is therefore $\mathcal{O}(\Gamma \cdot n + \Gamma \cdot m)$.

Next we analyze the running time complexity, communication complexity and rounds complexity of the secure computation part of the scheme. The entire secure computation is wrapped in a loop over the records in A', which has the same size as the input A. Within each iteration, there is another for loop over the constant Γ. Each iteration of Γ includes a single $\mathsf{Oram.access}$ and the addition of two m-sized tuples. The latter operation is free within our semi-honest attack model. The Square-Root $\mathsf{Oram.access}$ has an amortized secure computation complexity and communication complexity of

$$\mathcal{O}\left(\sqrt{|\mathsf{bk}| \log^3 |\mathsf{bk}|}\right)$$

and a $\mathcal{O}(\log|\mathsf{bk}|)$ round complexity, where $|\mathsf{bk}|$ denotes the amount of blocks stored in the ORAM. After Γ iterations, the procedure $\mathsf{GetTopMatches}$ performed.

The $\mathsf{GetTopMatches}$ procedure performs $\sum_i^{\log_2 \frac{m}{x}} \frac{m}{2^i} = m - x$ secure comparisons. Since $m \gg x$, the procedure results in a running time and communication cost of $\mathcal{O}(m)$. Due to the parallel nature of the while loop iteration, the round complexity scales only with the amount of while loop iterations which is $\mathcal{O}\big(\log \frac{m}{x}\big)$.

The full secure computation and communication complexity is

$$n \times (\Gamma \times \sqrt{|\mathsf{bk}| \log^3 |\mathsf{bk}|} + m),$$

with the dominant factor being $\mathcal{O}(n \times m)$.

The round complexity is $n \times (\Gamma \times \log|\mathsf{bk}| + \log m)$. We simplify this to $\mathcal{O}(n \log m)$ as $|bk| \leq m$ assuming Bob does not hold empty blocks. The resulting complexity figures are given in Table I.

F *Optimizations*

Square-Root ORAM Refresh One major running time optimization we can make is to run the Square-Root ORAM refresh operation in a multi-threaded fashion. Recall from Section III.C that we need to refresh an ORAM after a period of T accesses. This operation is the most expensive operation of the ORAM, as it requires a secure shuffle of the underlying memory. Note however that the ORAM $\langle blocks_\gamma \rangle$ for every blocking key bk is queried the exact same amount of times. Therefore, each ORAM will need to refresh during the same iteration of the loop in line 15. Performing the refresh in parallel with a separate thread per ORAM significantly improves the running time up to a factor of Γ.

VI SECURE LSH-BASED BLOCKING

As described in our model of blocking, we focus on blocking techniques which allow parties to independently and locally produce blocking keys. The blocking techniques *Token Blocking*, *Standard Blocking* and *Locality Sensitive Hashing* fit this definition. These techniques are briefly elaborated in Section II. Token-based techniques has been shown to have a poor reduction ratio [6]. Standard Blocking also generally tends to underperform compared to other blocking techniques [31, 34]. In this paper we therefore focus on Locality Sensitive Hashing.

A *Locality Sensitive Hashing*

Locality Sensitive Hashing (LSH) is a technique introduced in [17] to solve the *nearest neighbor* problem. This problem suffers from the curse of dimensionality, in that finding the nearest neighbor of a point in a high dimensional space quickly becomes infeasible using pair-wise comparisons for all dimensions. LSH is a tool to solve this problem approximately, by hashing items using a distance sensitive hashing function such that items that are spatially close produce the same hash. Hashing is generally a fast operation, making it an attractive solution for blocking.

LSH-based blocking is a blocking technique used in both the record linkage [31] and privacy-preserving record linkage literature [11]. The idea is to hash a record using a LSH function to place it into a block. Record pairs that are placed into the same block are regarded as candidate pairs. The records are hashed using multiple hashing functions from the same family to place them in multiple blocks, to increase the probability of similar records colliding for at least one hashing function.

Of particular interest in this paper are the LSH-based approaches which utilize binary vector representations of records as input, as they work well in a Secure Multi-Party Computation setting. The usual way to generate such representations is through adding the N-grams of a record to a Bloom Filters [6, 22] as described in Section III.D. Two common LSH families on binary input are Hamming LSH (H-LSH) and Jaccard LSH. We focus on the former, as it has been shown that Hamming LSH outperforms Jaccard LSH in terms of running time and recall [6, 22].

Definition VI.1 (Hamming distance)**.**

$$\begin{aligned} \mathsf{Hamming} : &(\{0,1\}^l, \{0,1\}^l) \to \mathbb{Z}_K \\ &(x, y) \mapsto |x \otimes y|, \end{aligned}$$

where $\otimes$ *denotes the bit-wise* XOR operation.

The family of Hamming LSH functions is sensitive to the hamming distance, meaning that a pair of elements with a small hamming distance is more likely to be hashed to the same block. In this family, a base hash function h samples uniformly at random a bit from the input.

Definition VI.2 (Hamming LSH (H-LSH))**.** *Let $S = \{0,1\}^l$. The family of Hamming LSH functions is defined as the set*

$$\mathcal{H} : \{h : S \to \{0,1\} : h(x) = x[i],\ i \in \mathbb{Z}_l\},$$

where $x[i]$ denotes the i'th bit in x.

The family $\mathcal{H}$ is (r,p)-sensitive for the hamming distance, such that any $h \in \mathcal{H}$

$$\begin{aligned} &\textit{if } \mathsf{Hamming}(x,y) \leq r \\ &\textit{then } \Pr[h(x) = h(y)] \geq p \quad \forall x, y \in S \\ &\textit{where } p = 1 - \frac{r}{l}. \end{aligned}$$

B *Frequent Pairs Scheme*

For the specific implementation of LSH-based blocking, we use the method by Karapiperis and Verykios [21], which is to our knowledge the state of the art in LSH-based blocking. They introduce the Frequent Pairs Scheme (FPS), which makes use of Hamming LSH hashing functions to generate collisions between pairs of records, and describe the optimal parameters for their scheme.

Given the Hamming LSH family $\mathcal{H}$, to generate blocking identifiers of κ bits we pick a composite hash function H consisting of κ base hash function as the blocking key function:

$$H = h_1 \in_R \mathcal{H} || \ldots || h_\kappa \in_R \mathcal{H}$$

The hashing table generated by applying the hashing function over the entire dataset generates the blocked dataset. The hashing function H is thus analogous to the notion of a blocking key.

To increase the recall of this method, authors [22, 21] redundantly use Γ blocking keys. Generating multiple blocking keys for each record increases the probability that a truly matching pair will collide for one of the blocking keys. We denote the blocking key for iteration $\gamma \in \{1, \ldots, \Gamma\}$ as $\mathsf{bk}_\gamma = H_\gamma$.

LSH-based methods thus employ multiple blocking keys to achieve a high recall. The negative consequence of this practise is a reduced precision as more false positives are generated. Therefore, the authors [21] introduce the notion of *collisions* and filter out only those candidate pairs that have a higher-than-average number of hash collisions for the desired distance threshold.

Parameters The quality of the FPS scheme is determined by three parameters, κ, δ and θ. The parameter κ, which has already been introduced, specifies the amount of bits of which the blocking identifiers should consist. The parameter θ specifies the fault tolerance of the scheme. Specifically, it specifies the maximum amount of bits which are expected to differ between two Bloom Filters of two records which represent the same entity. The higher this distance, the more typographic errors can be caught by the scheme. Lastly, δ is the confidence parameter which bounds from above the failure probability of two Bloom Filters with distance θ missing the collision threshold.

The mentioned collision threshold will serve as the threshold t as defined in Section IV.B. Both the threshold t and the number of blocking keys Γ are derived from κ, δ and θ (we refer to [21] for details).

VII RESULTS

This section describes the experimental setup used to test the performance of Algorithm 1 using the LSH-based blocking technique of Karapiperis and Verykios [21] introduced in the previous section, and reports the results.

The results are discussed in Section VIII.

A *Benchmarking setup*

The secure computation step of the scheme is implemented using the high-level interface of the MP-SPDZ Framework. The implementation makes use of multi-threading were applicable to speed up the computation. The compilation step which compiles the high-level code into 'secure assembly' is not included in the benchmarks. As explained in Section V.C, the 'semi2k' cryptographic protocol was used to run the assembly. The computation was compiled using a ring-size of 64 bits, a computational security level of 128, a statistical security level of 40, the default settings for MP-SPDZ, and edaBit enabled. The computation is also compiled with the '–invperm' flag, which enables the inverse permutation operation used by Sqrt-ORAM [48]. The execution spawns two processes, one for each party. Both parties run on the same machine, meaning the computations are not bound by bandwidth limitations. The running times reported in the results are the combined running time of both the offline and the online phases of the secure computation.

The benchmarks were run on virtual machines deployed in the Google Compute Engine. The virtual machines were equipped with 32GB of RAM and 8-vCPUs (4 physical cores) of type C2, the compute optimized CPU family of the platform. The specific model is Intel Xeon Gold 6253CL Processor running at 3.1GHz base frequency and 3.8GHz boost frequency.

B *Blocking Quality*

In this section we verify whether the quality of the blocking solution is in line with expectations. We describe the metrics relevant to blocking, followed by the benchmarking setup used to test the blocking quality.

1 *Metrics*: The quality of a blocking solution can be measured using the three standard metrics of the blocking literature [3].

The first metric is the Pairs Completeness (PC). This metric measures the ratio of truly matching pairs found by the blocking method, i.e. the recall. It is defined as $PC = \frac{|\tilde{\mathcal{M}} \cap M|}{|M|}$, where $\tilde{\mathcal{M}}$ is the set of candidate pairs

generated by the blocking solution, and M is the set of truly matching pairs. A low PC results in subsequent poor record linkage performance, as pairs that would be matched by RL are not placed in the same blocks. The second metric Reduction Ratio (RR) is defined as $RR = 1.0 - \frac{|A \times B|}{|\tilde{\mathcal{M}}|}$. The Reduction Ratio captures the reduction in the comparison space of record linkage, where a high reduction ratio means a highly reduced space. The metric is only relevant for blocking solutions to measure their effectiveness as a preprocessing step to improve performance of the record linkage step (see Figure 4).

The third metric is the Pairs Quality (PQ). It is defined as $PQ = \frac{|\tilde{\mathcal{M}} \cap M|}{|\tilde{\mathcal{M}}|}$. The Pairs Quality reflects the precision of the blocking solution in finding truly matching pairs.

2 *Benchmark*: To test the blocking quality we use the same experimental setup as in Stammler et al. [42]. Alice and Bob each sample a dataset of a 10.000 records, where A and B overlap in exactly 60% of the records, i.e. A and B both contain 6.000 identical records. The datasets have the attributes 'First Name', 'Surname', 'Birth date' and 'SSN'. In their comparison, Stammler et al. [42] use the non-free 'Mockaroo' service to generate a synthetic dataset of 50.000 records from which the required records for each experiment are sampled. In our experiments, we use a random subset of 50.000 records of the North Carolina Voter Registration Database [30]. This results in similar, but not identical datasets.

Next, the database of Bob is perturbed. Each field is probabilistically perturbed by two permutations, either a deletion of a random character or the exchange of two random characters. Each permutation is applied with equal probability, such that the total probability of a permutation of a field is 40%.

The fuzziness parameter θ used to generate the blocks was optimally set to 440. This parameter sets the maximum expected bit distance between two Bloom Filters representing the same entity. Given the expected change in bits for a deletion (30) and for a transposition (80) according to [21], the Bloom Filter distance for a true positive pair should be at most $4 \times (30 + 80) = 440$ (the maximum amount of permutation per field is known to be a single deletion and a single transposition).

The blocking quality was benchmarked using various settings of the parameters κ and δ. The results are shown in Table II. The general observation is that the number of false negatives is very low for $\delta \leq 0.001$, resulting in a near optimal Pairs Completeness (PC). The Reduction Ratio (RR) of $0,9375$ is high and translates into a large search space reduction, but is lower than that in [21]. Similarly, the Pairs Quality (PQ) is much smaller than in [21] due to the high number of false positives.

C *Running Time*

Algorithm 1 has four parameters, $|A|$, $|B|$, BK and x. The parameters of interest are the dataset sizes $|A|$ and $|B|$ as these are the parameters we wish to scale. As shown in the the complexity analysis, the number of blocking groups $\Gamma = |BK|$ also influences the running time. However, this parameter is set independently of $|A|$ and $|B|$ according to the blocking quality desired for the use case.

TABLE II

The quality of the blocking output for various values of κ and δ with x fixed to 625. The results are snapshots of a single execution of the Algorithm.

κ	δ	Time (s)	TP	FN	PC	PQ	RR
6	0,01	24907	5954	44	0,9927	0,00095	0,9375
7	0,01	35727	5989	9	0,9985	0,00096	0,9375
8	0,01	57037	5991	7	0,9988	0,00096	0,9375
6	0,001	35324	5991	8	0,9987	0,00096	0,9375
7	0,001	51543	5994	5	0,9992	0,00096	0,9375
8	0,001	86455	5995	3	0,9995	0,00096	0,9375
6	0,0001	47060	5995	2	0,9997	0,00096	0,9375
7	0,0001	68075	5990	4	0,9993	0,00096	0,9375
8	0,0001	110198	5996	3	0,9995	0,00096	0,9375

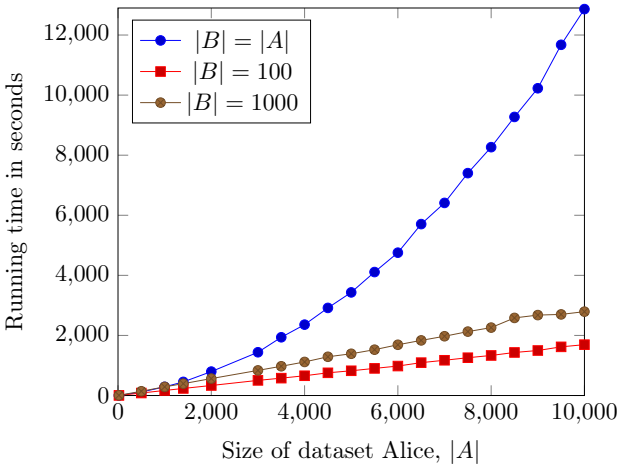

Fig. 5. Running Time for varying values of $|A|$ and $|B|$

To benchmark the running time, $|A|$ is varied between 1 and 10.000 in increments of 500. For each value of $|A|$, timings are reported for three different values for $|B|$, namely $|B| = |A|$, $|B| = 100$, and $|B| = 1000$.

The results of this experiment are shown in Figure 5. The graph shows that the experimental results are in line with the complexity analysis. When $|B|$ is set to $|A|$, the running time scales with $|B| \times |A|$. When $|B|$ is fixed, the running time scales linearly with $|A|$.

D *Parameter x*

To confirm that the parameter x has a negligible effect on the computational complexity, as claimed in the complexity analysis, we run several benchmarks for x with all other parameters being equal. Figure 6 shows the running time plotted against x. The graph shows the running time increasing slightly for increasing values of parameter x. This is due to the fact that the complexity analysis posed the assumption that $x \ll |B|$, whereas the graph plots much larger values for x.

Furthermore, parameter x determines the size of the result set $\tilde{\mathcal{M}}$, and thus influences the blocking quality metrics. To analyze the effect of parameter x on the blocking quality, we must first differentiate between false negatives resulting from the choice of parameter x and

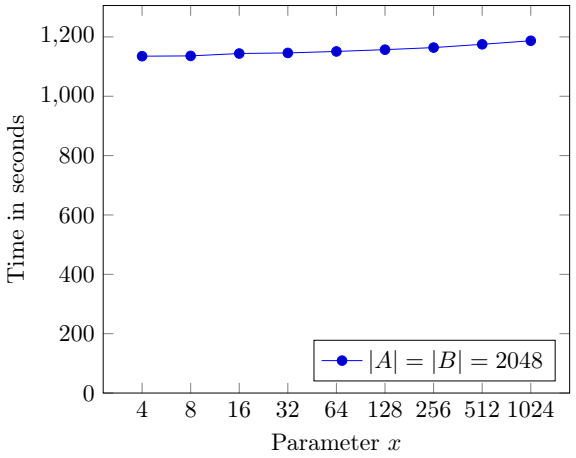

Fig. 6. Running times for different values of parameter x. The benchmarks use parameters $\kappa = 6, \delta = 0.001, \theta = 440$.

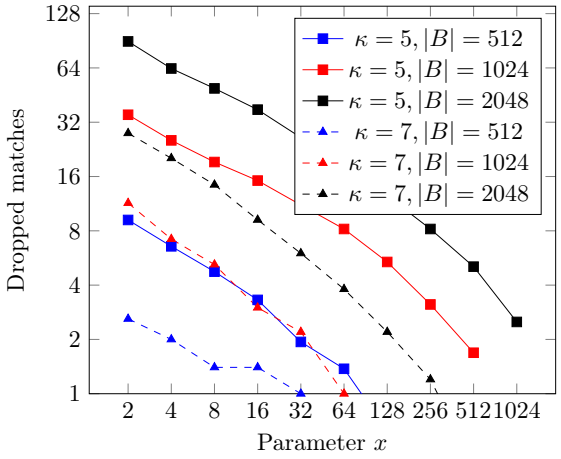

Fig. 7. The amount of dropped matches for varying values of x over the same dataset. The experiment was run using different values of κ. The remaining parameters were set to $|A| = |B| = 2048, \delta = 0.001, \theta = 440$.

the false negatives resulting from the failure probability of Locality Sensitive Hashing. The former set will be called the *dropped pairs*, i.e. pairs that didn't fit into the result set. The latter set is called the set of *missed pairs*. For parameter x we are interested in the set of *dropped pairs*.

We run several benchmarks with varying values of parameter x. The dataset sizes $|A|$ and $|B|$ were set to 2048, allowing for a wide range of values for x to be tested. The blocking quality parameters θ and δ were set such that the amount of matches missed is negligible. Each benchmark was run five times, gathering the average and the standard deviation.

Figure 7 shows the results of this experiment. Note that both axis are logarithmic scales. The graph shows a strong correlation between the parameter x and the number of dropped matches. Furthermore, the graph shows that for a higher value of κ, the number of dropped matches is lower overall.

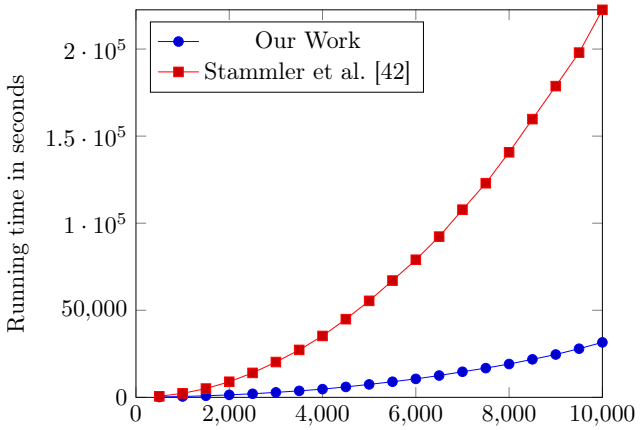

Fig. 8. Comparison of running times between our work and that of Stammler et al. [42].

E *Comparison to Stammler et al. [42]*

In order to show that the blocking solution indeed results in a practical decrease in running time, we compare the running time of our blocking solution to the record linkage solution of Stammler et al. [42]. Do note that in a real-world scenario a secure record linkage step would be needed after the secure blocking step to achieve the same functionality. However, the record linkage step after blocking is only computed on a greatly reduced subset of records.

To compare the solutions, we benchmarked both solutions (using the original code for Stammler et al. [42]) on the same machine described in Section VII.A. Both solutions were compiled for 32-bit computation with a computational security level of 128 bits. Both solutions were compiled in release mode for optimal performance. Both parties in a computation run on the same machine, such that no bandwidth restrictions are imposed.

For our solution, we chose to use the parameters $\kappa = 6, \theta = 440, \delta = 0.001$, which are the smallest parameters that would result in near-optimal blocking performance on a heavily permuted dataset as described in Section VII.B.

For the work of Stammler et al. [42], the GMW/A version of their solution is used which showed the best performance in their paper. For all attributes, the 'dice' comperator is used, which uses bloom filters to fuzzy match strings, the same as in this paper.

The solutions were benchmarked under varying dataset sizes ranging from 500 to 10000 in increments of 500. The records in the datasets have four string attributes. The results are shown in Figure 8.

VIII DISCUSSION

A *Blocking Quality*

The goal of blocking is to reduce the number of comparisons needed in the matching step (the next step in a full record-linkage pipeline as depicted in Figure 4). As such, blocking should have a high Reduction Ratio and Pairs Completeness. A high *PC* ensures that the quality degradation of the matching step is negligible. Every false

negative produced by the blocking step will consequently be missed in the matching step. Furthermore, a high RR is needed for the combination of the blocking step and matching step to run faster than a standalone matching step.

As observed in Section VII, the blocking solution performs well. This is expected, as our Secure Blocking solution implements the blocking technique of Karapiperis and Verykios [21]. For a deeper analysis of the impact of κ, δ and θ on the blocking quality, we refer to this paper. Instead, we will point out the differences in results between our solution and that of [21].

Though our solution uses the technique of [21] to block the datasets of Alice and Bob, the decision on which records to include in the result set $\tilde{\mathcal{M}}$ differs in our implementation, as the method in [21] does not translate well into the context of secure computation.In our implementation, the procedure GetTopMatches takes care of this step.

First of all, instead of filtering out record pairs with exactly t or more collisions, the procedure, for a single record $a \in A$ aims to return the set of exactly the top x records pairs $\{(a,b) : b \in B\}$ ranked according to their number of collisions. Setting x such that $x > |\{(a,b) : \mathsf{collisions}(a,b) \geq t, b \in B\}|$, guarantees false positives. Idem if x is smaller, false negatives are guaranteed.

The number of false positives is thus fixed to $n \times x - TP$. This directly impacts the RR and PQ. The results indicate that a good albeit lower RR is achieved compared to [21]. The PQ metric is more sensitive to the amount of false positives and thus is much smaller in our results. The results show that although that our implementation significantly reduces the comparison space, it produces more false positives than [21].

Secondly, as mentioned in Section V.D, the procedure GetTopMatches does not guarantee correctness.

The incorrectness stems from the fact that the procedure only performs a maximum of m secure comparisons to determine the top x counters in a vector of m counts. To illustrate, consider the vector $[2, 3, 1, 4]$. This vector is reduced to

$$[2, 3] \overset{?}{>} [1, 4] = [2, 4]$$

after one iteration of GetTopMatches, causing the 3 to be missed.

Upon analyzing the debugging information, it becomes clear that *all* false negatives generated in the benchmarks of Table II are in fact caused by the procedure GetTopMatches. The absence of *missed pairs* is in line with expectations, as true matching record pairs often have a distance lower than that of θ, resulting in a failure probability much smaller than δ. To verify, the same benchmarks were ran using a much smaller value $\theta = 176$, which did generate *missed pairs*.

Overall, our implementation results in a high RR and a high PC, but sacrifices PQ to achieve a faster secure computation.

B *Parameter x*

To reduce the chance of *dropped pairs*, the parameter x must be set generously. In terms of running time, this is not an issue. Figure 6 confirms the complexity analysis that parameter x has little influence on the running time. However, a larger x directly translates to a lower Reduction Ratio, which reduces the effectiveness of the blocking solution.

The experiment in Section VII.D gives us insight into the appropriate value for x (results depicted in Figure 7). We observe that x must scale with both the size of the dataset $|B|$, and with κ, to keep the amount of dropped pairs to a minimum. This first is straightforward. The procedure GetTopMatches collects the top x pairs from a vector of $|B|$ pairs. As the ration between x and $|B|$ becomes smaller, the chance that a valid candidate pair is 'pushed out' increases.

Second is κ. Recall that κ represents the amount of bits of a block identifier, thus determining the amount of blocks are contained in a single blocking table. Karapiperis and Verykios [21] note in their paper that κ must be set sufficiently large for a decent Pairs Quality. This is because the chance that a pair of dissimilar records finds itself in the same block increases as the amount of blocks decreases. In our setup, a smaller k not only indirectly reduces the PQ, but also directly the PC. This is because pairs of dissimilar records have a higher chance of generating t or more collisions, thus making them compete with true positive pairs for a spot in the vector produces by GetTopMatches.

The finding that κ must scale with $|A|$ and $|B|$ to achieve good blocking performance is in line with the earlier conclusions of [20, 21]. In our implementation however, there is an additional degradation in not only PQ but also PC due to the impact of κ on GetTopMatches. However, as can be seen Table II, near-optimal PC can be achieved on datasets of sizes in the order of 2^{13} for values of κ as low as 6.

C *Running Time*

There are two paths to take when designing a fast blocking solution. Either the algorithm must have a better algorithmic complexity than $|A| \times |B|$ or the algorithm must perform a very cheap comparison, ideally both. In the context of *secure record linkage*, we focus on the complexity and computational cost of the secure computation as this step dominates the running time.

In our solution, we aim to construct a cheap comparison by limiting the work that needs to be done in the secure computation. To this end, we limit our scope to independent blocking techniques, such that computation of the blocked datasets can be performed locally outside of a secure computation. What is left for the secure computation phase is only the comparison of blocking identifiers. Intuitively, this should be faster than running a more involved matching step.

Table II includes the running times in seconds for various benchmarks. The main result is that we managed to block two datasets of sizes $|A| = |B| = 10000$ in under

ten hours $\kappa = 6, \delta = 0.001$, including both the offline and online phase of the computation. Although this is orders of magnitude slower than a non-secure computation, the running times are small enough for some real-world uses on smaller datasets.

One promising use-case is blacklist checking. In such a setup, one party has a blacklist with a limited number of entries. Another party with a much larger dataset could check whether any of its records are contained in the blacklist. In general, use-cases where one dataset is much smaller than the others would suit well. In these scenarios, the run-time scales with $|A| = n$.

In terms of computational complexity, we can conclude from Figure 5 that our solution scales with $\mathcal{O}(n \times m)$ as predicted by our complexity analysis.

The main issue that prevents our design from reaching a smaller running time complexity has to do with the block length. In our current design we fixed the block-length to m. This length is used to store a single bit for each record $b \in B$, indicating its presence in the block. This allows us to efficiently count the collisions of a record of Alice a with each $b \in B$. However, in the end we must perform m secure comparisons to filter out the candidate record pairs. If we wish to perform less computation (asymptotically) for a single record a of Alice, the blocks retrieved from Bob should be smaller in length than m. This introduces a new problem. Now the blocks can no longer store a single bit for each $b \in B$ and must instead store a smaller number of integers, namely the indexes of Bob's records which are contained in the block. This complicates the process of counting the collisions. We would then have to introduce a new ORAM to be able to count the collisions based on the secret indexes retrieved from a block of Bob. We would end up with a complexity in the form

$$n \times \Gamma \times (\mathsf{Oram.read} + l \times \mathsf{Oram.write}),$$

i.e. for every blocking key, we read a block of length l. For every index in the block, we would need an ORAM `write` instruction to increase the collision counter for this index. Problematically, if we wish to reduce the average block length $l = \frac{n}{|\mathsf{bk}|}$ (disregarding the deviation, which also needs to be taken into account), we would have to increase the number of blocks $|\mathsf{bk}| = 2^\kappa$. In turn, we would need to increase Γ as Γ scales with κ.

The problem here is thus that to reduce the complexity, we need to reduce the block length by increasing κ which in turn increases the needed amount of blocking keys to retain the blocking quality. A blocking scheme with a completely independent number of blocking keys (likely not a hashing based method) might not suffer from this limitation.

D *Comparison to Stammler et al. [42]*

In Section VII we also compared the running time of our blocking method to the record linkage solution of Stammler et al. [42]. As their work is the only one that provides a complete record linkage solution built as a secure computation, this is currently the only work that would benefit from a secure blocking step. Therefore, it is interesting to analyze whether our solution would actually speed up this method.

The results in Figure 8 clearly show that although both solutions scale quadratically, the running time of our solution is much smaller. Fitting both curves to the quadratic function $a + C \times n^2$, we get that our solution achieves roughly a 7x fold reduction in running time. The benchmarks were ran assuming a heavily permuted dataset, which might not be necessary for real-world use. Reducing the fuzziness parameter of θ our solution would furthermore significantly reduce the running time, whereas this scaling is not applicable in [42].

It should however be noted that the secure computations were implemented using different frameworks. While our work uses MP-SPDZ, [42] uses the ABY framework. For a more definitive benchmark, both solutions would have to be implemented in the same framework. Furthermore, the benchmark should compare the running times of record linkage with and without blocking, not only stand-alone blocking.

One limitation of our secure blocking solution in comparison to the record linkage solution [42], is that our solution only considers string attributes. The solution in [42] also considers integer attributes, for which a different distance metric is implemented.

IX CONCLUSION

In this work, we have presented the first *secure* blocking solution. With this solution, we make a step towards realizing a full record-linkage pipeline as a secure computation.

We say our solution is secure as it reveals no information other than the result set $\tilde{\mathcal{M}}$ given the CDH assumption. The results in this paper were all generated under the assumption of the 'semi-honest' or 'honest-but-curious' attacker model. This model assumes that the parties in the protocol will not deviate from the protocol. As this solution was built using the MP-SPDZ framework, it can easily be compiled to the 'malicious' attacker model, although at a significant hit to run-time performance. The framework also implements protocols based on different security assumptions. Thus, if post-quantum security is desired, the algorithm could be compiled to use oblivious transfers based on the LWE assumption.

To achieve secure blocking, we have taken the idea of Stammler et al. [42] to employ Oblivious RAM (ORAM) for implementing a secure blocking solution, and shown its feasibility. The solution produces very low false negative rates for large datasets in the context of secure computation using the blocking technique of Karapiperis and Verykios [21], meaning the quality of the result set is applicable to real-world use-cases.

Furthermore, the solution presented in this paper is not limited to the blocking technique of [21]. Any blocking technique which allows parties to locally and independently partition their dataset could be used to implement a secure blocking solution. As such, this paper more generally demonstrates the applicability of ORAM for this problem.

Our benchmarks comparing the running time of our solution to the record linkage solution of Stammler et al. [42] show that our solution runs much faster on datasets of the same size, and thus our solution can be used to reduce the overall running time of a record linkage solution.

Future Work In this paper we chose to implement the Sqrt-ORAM construction by Zahur et al. [48] as building block for our secure blocking solution. A more recent work published by Doerner and Shelat [4] presents the FloRAM construction. This construction requires plaintext local computation, which does not fit the MP-SPDZ framework well. However, the FloRAM construction does not require an expensive initialization procedure, requires asymptotically less secure computation and outperforms all previous construction in terms of concrete running-time for a large range of parameters. This construction would likely result in faster running times.

LIST OF SYMBOLS

- δ The confidence parameter, which gives that a pair of records with a bloom filter hamming distance equal to θ fails to reach the required the collision threshold..
- κ The bit-length of a blocking identifier. A bit-length of k results in 2^κ blocks in a single blocking table..
- θ The fuzziness parameter specifying the expected amount of bits to differ between the bloom filters of a pair of records which represent the same entity..
- m Size of Bob's dataset, $|B|$.
- n Size of Alice's dataset, $|A|$.
- t The collision threshold, i.e. the amount of blocking tables on which a pair of records should collide before being counted as a candidate record pair. This value is derived from δ, κ and θ..
- x The amount of candidate record pairs to generate for a single record. Used in the quadratic complexity version of the Algorithm..

REFERENCES

[1] *A Pragmatic Introduction to Secure Multi-Party Computation*. URL: https://securecomputation.org/ (visited on 03/09/2022).

[2] Burton H. Bloom. "Space/Time Trade-Offs in Hash Coding with Allowable Errors". In: *Communications of the ACM* 13.7 (July 1, 1970), pp. 422–426. ISSN: 0001-0782. DOI: 10.1145/362686.362692. URL: http://doi.org/10.1145/362686.362692 (visited on 02/17/2022).

[3] Peter Christen. "A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication". In: *IEEE Transactions on Knowledge and Data Engineering* 24.9 (Sept. 2012), pp. 1537–1555. ISSN: 1558-2191. DOI: 10.1109/TKDE.2011.127.

[4] Jack Doerner and Abhi Shelat. "Scaling ORAM for Secure Computation". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. New York, NY, USA: Association for Computing Machinery, Oct. 30, 2017, pp. 523–535. ISBN: 978-1-4503-4946-8. DOI: 10.1145/3133956.3133967. URL: https://doi.org/10.1145/3133956.3133967 (visited on 03/10/2022).

[5] Halbert L. Dunn. "Record Linkage". In: *American Journal of Public Health and the Nations Health* 36.12 (1946), pp. 1412–1416.

[6] Elizabeth Ashley Durham. "A Framework for Accurate, Efficient Private Record Linkage". In: (Apr. 9, 2012). URL: https://ir.vanderbilt.edu/handle/1803/11417 (visited on 04/06/2022).

[7] Cynthia Dwork. "Differential Privacy". In: *Automata, Languages and Programming*. Ed. by Michele Bugliesi et al. Berlin, Heidelberg: Springer, 2006, pp. 1–12. ISBN: 978-3-540-35908-1. DOI: 10.1007/11787006_1.

[8] Aleksander Essex. "Secure Approximate String Matching for Privacy-Preserving Record Linkage". In: *IEEE Transactions on Information Forensics and Security* 14.10 (Oct. 2019), pp. 2623–2632. ISSN: 1556-6013, 1556-6021. DOI: 10.1109/TIFS.2019.2903651. URL: https://ieeexplore.ieee.org/document/8662592/ (visited on 02/03/2022).

[9] Ivan P. Fellegi and Alan B. Sunter. "A Theory for Record Linkage". In: *Journal of the American Statistical Association* 64.328 (Dec. 1, 1969), pp. 1183–1210. ISSN: 0162-1459. DOI: 10.1080/01621459.1969.10501049. URL: https://www.tandfonline.com/doi/abs/10.1080/01621459.1969.10501049 (visited on 02/16/2022).

[10] Craig Gentry et al. "Optimizing ORAM and Using It Efficiently for Secure Computation". In: *Privacy Enhancing Technologies*. Ed. by Emiliano De Cristofaro and Matthew Wright. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2013, pp. 1–18. ISBN: 978-3-642-39077-7. DOI: 10.1007/978-3-642-39077-7_1.

[11] Aris Gkoulalas-Divanis et al. "Modern Privacy-Preserving Record Linkage Techniques: An Overview". In: *IEEE Transactions on Information Forensics and Security* 16 (2021), pp. 4966–4987. ISSN: 1556-6021. DOI: 10.1109/TIFS.2021.3114026.

[12] Oded Goldreich and Rafail Ostrovsky. "Software Protection and Simulation on Oblivious RAMs". In: *Journal of the ACM* 43.3 (May 1, 1996), pp. 431–473. ISSN: 0004-5411. DOI: 10.1145/233551.233553. URL: http://doi.org/10.1145/233551.233553 (visited on 03/04/2022).

[13] S. Dov Gordon et al. "Secure Two-Party Computation in Sublinear (Amortized) Time". In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. CCS '12. New York, NY, USA: Association for Computing Machinery, Oct. 16, 2012, pp. 513–524. ISBN: 978-1-4503-1651-4. DOI: 10.1145/2382196.2382251. URL: https://doi.org/10.1145/2382196.2382251 (visited on 03/07/2022).

[14] Xi He et al. "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*. CCS '17. New York, NY, USA: Association for Computing Machinery, Oct. 30, 2017, pp. 1389–1406. ISBN: 978-1-4503-4946-8. DOI: 10.1145/3133956.3134030. URL: https://doi.org/10.1145/3133956.3134030 (visited on 03/05/2022).

[15] Ali Inan et al. "A Hybrid Approach to Private Record Linkage". In: *2008 IEEE 24th International Conference on Data Engineering*. 2008 IEEE 24th International Conference on Data Engineering. Apr. 2008, pp. 496–505. DOI: 10.1109/ICDE.2008.4497458.

[16] Ali Inan et al. "Private Record Matching Using Differential Privacy". In: *Proceedings of the 13th International Conference on Extending Database Technology*. EDBT '10. New York, NY, USA: Association for Computing Machinery, Mar. 22, 2010, pp. 123–134. ISBN: 978-1-60558-945-9. DOI: 10.1145/1739041.1739059. URL: https://doi.org/10.1145/1739041.1739059 (visited on 03/21/2022).

[17] Piotr Indyk and Rajeev Motwani. "Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality". In: *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*. STOC '98. New York, NY, USA: Association for Computing Machinery, May 23, 1998, pp. 604–613. ISBN: 978-0-89791-962-3. DOI: 10.1145/276698.276876. URL: https://doi.org/10.1145/276698.276876 (visited on 04/20/2022).

[18] Katie Irvine et al. "Real World Performance of Privacy Preserving Record Linkage". In: *International Journal of Population Data Science* 3 (Sept. 10, 2018). DOI: 10.23889/ijpds.v3i4.990.

[19] Alexandros Karakasidis and Vassilios S. Verykios. "Reference Table Based K-Anonymous Private Blocking". In: *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. SAC '12. New York, NY, USA: Association for Computing Machinery, Mar. 26, 2012, pp. 859–864. ISBN: 978-1-4503-0857-1. DOI: 10.1145/2245276.2245444. URL: https://doi.org/10.1145/2245276.2245444 (visited on 04/06/2022).

[20] Alexandros Karakasidis and Vassilios S. Verykios. "Secure Blocking + Secure Matching = Secure Record Linkage". In: *Journal of Computing Science and Engineering* 5.3 (2011), pp. 223–235. ISSN: 1976-4677. DOI: 10.5626/JCSE.2011.5.3.223. URL: https://www.koreascience.or.kr/article/JAKO201128762648380.page (visited on 04/05/2022).

[21] Dimitrios Karapiperis and Vassilios S. Verykios. "A Fast and Efficient Hamming LSH-based Scheme for Accurate Linkage". In: *Knowledge and Information Systems* 49.3 (Dec. 1, 2016), pp. 861–884. ISSN: 0219-3116. DOI: 10.1007/s10115-016-0919-y. URL: https://doi.org/10.1007/s10115-016-0919-y (visited on 04/07/2022).

[22] Dimitrios Karapiperis and Vassilios S. Verykios. "An LSH-Based Blocking Approach with a Homomorphic Matching Technique for Privacy-Preserving Record Linkage". In: *IEEE Transactions on Knowledge and Data Engineering* 27.4 (Apr. 2015), pp. 909–921. ISSN: 1558-2191. DOI: 10.1109/TKDE.2014.2349916.

[23] Marcel Keller. "MP-SPDZ: A Versatile Framework for Multi-Party Computation". In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security. Virtual Event USA: ACM, Oct. 30, 2020, pp. 1575–1590. ISBN: 978-1-4503-7089-9. DOI: 10.1145/3372297.3417872. URL: https://dl.acm.org/doi/10.1145/3372297.3417872 (visited on 05/04/2022).

[24] Marcel Keller and Peter Scholl. "Efficient, Oblivious Data Structures for MPC". In: *Advances in Cryptology – ASIACRYPT 2014*. Ed. by Palash Sarkar and Tetsu Iwata. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2014, pp. 506–525. ISBN: 978-3-662-45608-8. DOI: 10.1007/978-3-662-45608-8_27.

[25] Mehmet Kuzu et al. "A Constraint Satisfaction Cryptanalysis of Bloom Filters in Private Record Linkage". In: *Privacy Enhancing Technologies*. Ed. by Simone Fischer-Hübner and Nicholas Hopper. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2011, pp. 226–245. ISBN: 978-3-642-22263-4. DOI: 10.1007/978-3-642-22263-4_13.

[26] Ali Al-Lawati, Dongwon Lee, and Patrick McDaniel. "Blocking-Aware Private Record Linkage". In: *Proceedings of the 2nd International Workshop on Information Quality in Information Systems*. IQIS '05. New York, NY, USA: Association for Computing Machinery, June 17, 2005, pp. 59–68. ISBN: 978-1-59593-160-3. DOI: 10.1145/1077501.1077513. URL: https://doi.org/10.1145/1077501.1077513 (visited on 04/05/2022).

[27] Ibrahim Lazrig et al. “Privacy Preserving Probabilistic Record Linkage Without Trusted Third Party”. In: *2018 16th Annual Conference on Privacy, Security and Trust (PST)*. 2018 16th Annual Conference on Privacy, Security and Trust (PST). Aug. 2018, pp. 1–10. DOI: 10.1109/PST.2018.8514192.

[28] Ashwin Machanavajjhala et al. “L-Diversity: Privacy beyond k-Anonymity”. In: *ACM Transactions on Knowledge Discovery from Data* 1.1 (Mar. 1, 2007), 3–es. ISSN: 1556-4681. DOI: 10.1145/1217299.1217302. URL: https://doi.org/10.1145/1217299.1217302 (visited on 10/20/2022).

[29] Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. “Efficient Clustering of High-Dimensional Data Sets with Application to Reference Matching”. In: *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2000, pp. 169–178.

[30] “North Carolina Statewide Voter Registration”. In: (). URL: https://www.ncsbe.gov/results-data/voter-registration-data (visited on 10/22/2022).

[31] Kevin O’Hare, Anna Jurek-Loughrey, and Cassio de Campos. “A Review of Unsupervised and Semi-supervised Blocking Methods for Record Linkage”. In: *Linking and Mining Heterogeneous and Multi-view Data*. Ed. by Deepak P and Anna Jurek-Loughrey. Cham: Springer International Publishing, 2019, pp. 79–105. ISBN: 978-3-030-01872-6. DOI: 10.1007/978-3-030-01872-6_4. URL: https://doi.org/10.1007/978-3-030-01872-6_4 (visited on 04/08/2022).

[32] *Optimising Care by Encrypting Patient Data*. TNO. URL: https://www.tno.nl/en/focus-areas/information-communication-technology/roadmaps/data-sharing/optimising-care-by-encrypting-patient-data/ (visited on 03/23/2022).

[33] Rafail Ostrovsky and Victor Shoup. “Private Information Storage”. In: Conference Proceedings of the Annual ACM Symposium on Theory of Computing. 1997, pp. 294–303.

[34] George Papadakis et al. “Blocking and Filtering Techniques for Entity Resolution: A Survey”. In: *ACM Computing Surveys* 53.2 (Mar. 13, 2020), 31:1–31:42. ISSN: 0360-0300. DOI: 10.1145/3377455. URL: https://doi.org/10.1145/3377455 (visited on 04/05/2022).

[35] Thilina Ranbaduge et al. “Hashing-Based Distributed Multi-party Blocking for Privacy-Preserving Record Linkage”. In: *Advances in Knowledge Discovery and Data Mining*. Ed. by James Bailey et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 415–427. ISBN: 978-3-319-31750-2. DOI: 10.1007/978-3-319-31750-2_33.

[36] Sean M. Randall et al. “Privacy-Preserving Record Linkage on Large Real World Datasets”. In: *Journal of Biomedical Informatics* 50 (Aug. 2014), pp. 205–212. ISSN: 1532-0480. DOI: 10.1016/j.jbi.2013.12.003. pmid: 24333482.

[37] Fang-Yu Rao et al. “Hybrid Private Record Linkage: Separating Differentially Private Synopses from Matching Records”. In: *ACM Transactions on Privacy and Security* 22.3 (Apr. 26, 2019), 15:1–15:36. ISSN: 2471-2566. DOI: 10.1145/3318462. URL: https://doi.org/10.1145/3318462 (visited on 04/07/2022).

[38] *Regulation (EU) 2016/679 of the European Parliament and of the council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 96/46/EC (General Data protection Regulation)*. European Commission. Apr. 5, 2016. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:02016R0679-20160504 (visited on 03/23/2022).

[39] Monica Scannapieco et al. “Privacy Preserving Schema and Data Matching”. In: *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’07. New York, NY, USA: Association for Computing Machinery, June 11, 2007, pp. 653–664. ISBN: 978-1-59593-686-8. DOI: 10.1145/1247480.1247553. URL: https://doi.org/10.1145/1247480.1247553 (visited on 03/21/2022).

[40] Rainer Schnell. “Privacy-Preserving Record Linkage”. In: (2015).

[41] Rainer Schnell, Tobias Bachteler, and Jörg Reiher. “Privacy-Preserving Record Linkage Using Bloom Filters”. In: *BMC Medical Informatics and Decision Making* 9.1 (Dec. 2009), p. 41. ISSN: 1472-6947. DOI: 10.1186/1472-6947-9-41. URL: https://bmcmedinformdecismak.biomedcentral.com/articles/10.1186/1472-6947-9-41 (visited on 02/17/2022).

[42] Sebastian Stammler et al. “Mainzelliste SecureEpiLinker (MainSEL): Privacy-Preserving Record Linkage Using Secure Multi-Party Computation”. In: *Bioinformatics* (Sept. 1, 2020), btaa764. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btaa764. URL: https://doi.org/10.1093/bioinformatics/btaa764 (visited on 02/25/2022).

[43] Dinusha Vatsalan and Peter Christen. “Sorted Nearest Neighborhood Clustering for Efficient Private Blocking”. In: *Advances in Knowledge Discovery and Data Mining*. Ed. by Jian Pei et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2013, pp. 341–352. ISBN: 978-3-642-37456-2. DOI: 10.1007/978-3-642-37456-2_29.

[44] Xiao Wang, Hubert Chan, and Elaine Shi. "Circuit ORAM: On Tightness of the Goldreich-Ostrovsky Lower Bound". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. CCS '15. New York, NY, USA: Association for Computing Machinery, Oct. 12, 2015, pp. 850–861. ISBN: 978-1-4503-3832-5. DOI: 10.1145/2810103.2813634. URL: https://doi.org/10.1145/2810103.2813634 (visited on 03/13/2022).

[45] Xiao Shaun Wang et al. "SCORAM: Oblivious RAM for Secure Computation". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. CCS '14. New York, NY, USA: Association for Computing Machinery, Nov. 3, 2014, pp. 191–202. ISBN: 978-1-4503-2957-6. DOI: 10.1145/2660267.2660365. URL: https://doi.org/10.1145/2660267.2660365 (visited on 03/09/2022).

[46] Mohamed Yakout, Mikhail J. Atallah, and Ahmed Elmagarmid. "Efficient Private Record Linkage". In: *2009 IEEE 25th International Conference on Data Engineering*. 2009 IEEE 25th International Conference on Data Engineering. Mar. 2009, pp. 1283–1286. DOI: 10.1109/ICDE.2009.221.

[47] Andrew Chi-Chih Yao. "How to Generate and Exchange Secrets". In: *27th Annual Symposium on Foundations of Computer Science (Sfcs 1986)*. 27th Annual Symposium on Foundations of Computer Science (Sfcs 1986). Oct. 1986, pp. 162–167. DOI: 10.1109/SFCS.1986.25.

[48] Samee Zahur et al. "Revisiting Square-Root ORAM: Efficient Random Access in Multi-party Computation". In: *2016 IEEE Symposium on Security and Privacy (SP)*. 2016 IEEE Symposium on Security and Privacy (SP). May 2016, pp. 218–234. DOI: 10.1109/SP.2016.21.