# Using Fuzzy Vaults for Privacy Preserving Record Linkage

Xhino Mullaymeri
Department of Applied Informatics
University of Macedonia
Thessaloniki, Greece
dai16019@uom.edu.gr

Alexandros Karakasidis
Department of Applied Informatics
University of Macedonia
Thessaloniki, Greece
a.karakasidis@uom.edu.gr

## ABSTRACT

Approximate string matching has a variety of applications, one of them being record linkage, where records from different sources, without common unique identifiers, are matched. When these records refer to individuals, maintaining their privacy becomes a paramount requirement, leading us to develop privacy preserving record linkage methods. In this paper, we present a solution to the privacy preserving record linkage problem which is based on Fuzzy Vaults, having the advantage of being two-party based. A Fuzzy Vault is a cryptographic structure based on noise. Fuzzy Vaults have been previously used mainly in biometric applications. In this work, we explain our methodology in depth, providing detailed examples, we support our argument for privacy preservation by a solid discussion and provide extensive experimental results accompanied by a rigorous analysis, not only to indicate the stability and robustness of our method, but also to demonstrate that our setup manages to achieve superior results when compared to a baseline three party - based method.

## KEYWORDS

Fuzzy Vault, Privacy Preserving Record Linkage, Approximate String Matching

## 1 INTRODUCTION

In this digital era we are living, the number of agencies and corporations which store personal data has dramatically increased. A common need for these organizations to exchange information for individuals described by these data and perform analysis on them. As a data integration step, this would traditionally boil down to a database join in the case that common unique identifiers are shared such as social id or a driving license number. However, since this is not always the case, the next step is to exploit fields that, when combined may uniquely identify a record. What we have just described is commonly known in the literature as the *Record Linkage Problem*. Solving this problem, however, does not take into consideration that the privacy of the described individuals in these databases should be maintained. This constitutes the *Privacy Preserving Record Linkage* problem, where two parties aim to find common records without revealing any additional information to each other, apart from the matching ones. However, data are usually dirty and even common fields between the two parties' databases might diverge, quite often due to human error (i.e. typing errors during data entry). Since quite often such identifiers are string values as name, surname or address, one approach to overcome this complication is by using approximate string matching methods which should also consider privacy preservation.

To indicate the importance of the Privacy Preserving Record Linkage task, let us consider the COVID-19 pandemic we are experiencing and the need for tracking the contacts of confirmed cases, which undoubtedly constitutes a matter of great concern. Suppose that a verified case of the disease is an airline passenger. To ensure public health, all patient's close contacts should be tracked and informed that they might have been infected. Applying Privacy Preserving Record Linkage on databases from health agencies and airlines, authorities may locate and inform people who might have been exposed to the virus, without compromising privacy. In the same context, surveillance of quarantined cases can also be benefited. When someone is placed under house quarantine, his identifying data may be added on a public governmental database so as to be accessed by every agency which offers public services (accommodation, transport, entertainment). As these places are usually overcrowded, makes them places of possible outbreaks. Before providing any services, such providers should be able to check if their customers' names are on the quarantined cases list, protecting, in any case, the privacy of the individuals in all sites.

In this paper, we present in detail the operation and the under-the-hood implementation details of the two-party method for performing privacy preserving record linkage using Fuzzy Vaults briefly sketched in [18]. A Fuzzy Vault scheme [12] is a cryptographic structure where a secret is being "locked" with a key and in order to "unlock" it someone must use a key which is similar enough to the locking key. This "unlocking" phase of a Fuzzy Vault is by its nature approximate, so it suits very well our goal for approximate string matching. Originally, It has been used for biometric authentication algorithms and there are such implementations displayed in the literature [25]. Despite the work on biometric authentication, to the best of our knowledge, there have been no works utilizing Fuzzy Vaults for performing privacy preserving approximate string matching, focused on, but not limited to, Privacy Preserving Record Linkage operations.

As such, our contributions may be summarized as follows. First of all, we illustrate, for the first time, all the details of our novel methodology using Fuzzy Vaults for private approximate string matching so as to be utilized in a privacy preserving record linkage context. This methodology consists of a carefully designed workflow employing reference sets, non bijective mapping functions and noise addition. We provide detailed examples in order to allow the in-depth comprehension of our method's details and we support our argument of privacy preservation by a solid discussion. In terms of empirical evaluation, we provide extensive experimentation, not only for exhibiting our method's behavior but for proving its superiority against a baseline three party-based method in terms of overall performance, a fact with additional value considering that our fuzzy vault approach does not require a third party.

The rest of this paper is organized as follows. Section 2 presents works related to ours. In Section 3, there is the background for introducing the reader to our method, which is detailed in Section 4. Section 5 contains a discussion over the privacy properties of our method. In Section 6 we provide the empirical analysis of our method in order to demonstrate its robustness and stability.

Finally, in Section 7, we provide our conclusions and thoughts for future research.

## 2 RELATED WORK

Private approximate string matching techniques have been widely developed in the context of privacy preserving record linkage [26], being mostly based either on Secure Multiparty Computation (SMC), usually incurring additional communication costs, or on data perturbation, that often seem to be quite vulnerable to attacks [5].

In order to overcome some of the drawbacks and render SMC based techniques applicable as it comes to real world size datasets, hybrid methods have emerged. In [11] a hybrid method which combines differential privacy and homomorphic encryption is presented. Here, instead of sanitizing data, differential privacy is applied to ensure data privacy. Afterwards, data partitioning into blocks takes place. Finally, an SMC step based on homomorphic encryption occurs and returns the matching pairs. This method manages to partially improve the situation with the bottleneck of the limited size of data that SMC can endure.

Another hybrid method which is based upon blocking, utilization of differential privacy and on an SMC (homomorphic) phase is described in [16]. However, in this case the two parties compare their data with the help of a third one. Moreover, even recent techniques addressing this problem employ a third party [21], an approach that may be rendered impractical requiring the availability of such a third party.

In the last few years, there has been particular effort towards developing two-party based techniques [3]. One of the methods used to achieve this functionality is homomorphic encryption [9], which, however is a computationally expensive method [1] also liable to suffer certain attacks [10]. Another recent technique is based on SPDZ [15] which, however, may also be expensive in terms of communication and computation [3]. Finally, Garbled Circuits can offer a solution [2], but further evaluation is needed in terms of execution time, size and reusability [22].

To this end, we are presenting a solution to the two-party privacy preserving record linkage problem using Fuzzy Vaults. A Fuzzy Vault scheme may be used for privacy preserving matching, personal entropy systems, biometrics [12], [25] or, as indicated by the recent literature [20], for signature verification. Our method has been briefly presented in [18]. In this paper, we move one step further by providing a simple protocol for two-party privacy-preserving record linkage using Fuzzy Vaults, detailed examples, an in-depth privacy discussion, exhibiting the privacy characteristics of our method and, last but not least, an extended experimental evaluation, also including a comparison with a baseline method.

## 3 PRELIMINARIES

In this section, we define the problem we are solving and provide the required background for laying out our method. The notation used throughout this paper is summarized in Table 1.

### 3.1 Problem Formulation

Let us consider two data sources, called Alice ($A$) and Bob ($B$). *Privacy preserving record linkage* is the problem of identifying (linking) all pairs of their records that refer to the same real world entity, so that no more information is disclosed to either $A$, $B$ or any third party involved in the process besides the identifiers of the linked records.

Alice and Bob will most probably use different schemas in their databases. As such, they will have different attributes. Let $S^A$ be Alice's schema and $S^B$ be Bob's schema and let us assume that in these schemas $m$ of the attributes are common between the two sources forming a composite key. These attributes might be names, surnames, addresses and so on. As such, none of these on its own may comprise a unique identifier that can be used to identify a record. We refer to these attributes as *matching fields*. This composite key is used to determine when two records *match*, i.e., when they refer to the same entity. To determine when two records match, the respective attributes forming the composite key need to be compared. Considering that our data is often dirty, matching should rely on a similarity or distance function.

Let us consider a similarity function $F() \rightarrow [0..1]$ and a threshold $t > 0$. Given the composite keys $c^A$ and $c^B$ for records $r^A$ and $r^B$, respectively, we define the following matching function $G \rightarrow \{0, 1\}$:

$$G(c^A, c^B) = \begin{cases} 1, & \text{iff } F(c^A, c^B) \geq t \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

If $G(c^A, c^B) = 1$, then the pair $(r^A, r^B)$ is a match.

This process is the *matching process*. To preserve privacy, i.e., ensure *privacy preserving matching* (PPM), after the completion of this process, the only information revealed is the identifiers of the matched records. In our case, where we use Fuzzy Vaults for matching, Formula 1 will have as input Bob's key, $K^B$ corresponding to record $r^B$, and a Fuzzy Vault $V$ for Alice's $r^A$ and it will examine their absolute matching status using the method we describe in detail in Section 4, returning 1 when Bob successfully unlocks Alice's Fuzzy Vault with his and 0 otherwise. As such, the matching threshold is equal to 1.

### 3.2 Lagrange Interpolation

Lagrange interpolation [24] is the most common interpolation method used in the fuzzy vault literature. Considering a two-dimensional Euclidean space, and given a data set of n + 1 points: $(x_0, y_0), (x_1, y_2), \ldots, (x_n, y_n) \quad x_i \neq x_j \; \forall \; i, j \in \{0, 1, \ldots, n\}$ the approximation function using Lagrange interpolation is a polynomial of $n^{th}$ degree and is defined as shown in Equation 2:

$$P(x) = \sum_{i=0}^{n} y_i * L_{n,i}(x) \quad (2)$$

This method is called Lagrange interpolation because it uses Lagrange basis polynomials in order to interpolate the data set, as illustrated in Equation 3:

$$L_{n,i} = \prod_{j=0, i \neq j}^{n} \frac{x - x_j}{x_i - x_j} \quad (3)$$

By noticing the denominator of Equation 3 it is evident why $x_i \neq x_j \; \forall \; i, j$ should hold for the formula's abscissas.

### 3.3 Fuzzy Vault

A Fuzzy vault is a cryptographic construct used for secret sharing, introduced in [12] and, in brief, it works as follows. Alice, wants to protect a secret $S$ by locking it up with a key, which is a set of items $K^A$. This is the *Lock* phase. Bob, may have access to this secret only if he uses a key, consisting of a set of items $B$ is substantially similar to Alice's key. This is the *Unlock* phase. To assure privacy, noise is also injected during the Lock phase.

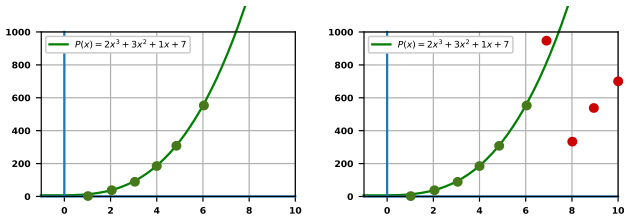



**Figure 1: Fuzzy Vault for locking 2317 into $P(x) = 2x^3 + 3x^2 + 1x + 7$. Left: Polynomial creation. Right: Fuzzy Vault combining genuine (green) and chaff (red) points.**

Next, we will further explain the details of this process and employ a running example to facilitate the reader. The original Fuzzy Vault construct was designed for numerical values, which we will use to illustrate its operation. However, as our work focuses on strings, we will lay out in later sections our methodology for achieving suitable transformations in order to adapt this concept for alphanumeric data.

*3.3.1 Lock Phase.* To illustrate the operation of a Fuzzy Vault, we will commence with the Lock phase and consider, as stated earlier, that Alice wants to lock her secret $S$ using her key, $K^A$. For this purpose, she uses a polynomial $P$ of degree $n$ so that $P$'s coefficients are, in fact, mappings of her secret $S$. Then, she uses the members of her key $K^A$ as x-coordinates for $P$ creating a collection of points $R$. As $K^A$ is a set, its elements are unique, thus the the x-coordinates of $R$'s points are going to be unique as well.

Let us now denote with $|K^A|$ the cardinality of $K^A$. Equation 4 illustrates the restriction that has to be satisfied with regard to the size of the polynomial used. This is due to Lagrange's interpolation method which requires $n + 1$ data points in order to return a polynomial of degree at most $n$ [7]. The key set $K^A$ holds the abscissas of the data points that Lagrange will later interpolate in the unlock phase.

$$|K^A| \geq n + 1 \tag{4}$$

Now, Alice will also create noise in order to protect the Fuzzy Vault's, thus offering privacy. To do so, she introduces a distraction set of points designated by $C$ and referred to as *Chaff points*. These points do not belong to the polynomials's curve. Their purpose is to confuse an attacker by refraining her from identifying the points in $R$ that have resulted from the Lock phase. As such, collection $C$ may be considered as the protection of the vault. The union of collections $C$ and $R$ comprised the vault $V$, or more formally: $\forall V, V = R \cup C$. In fact, $K^A$ is considered to be the key of the vault $V$ since it identifies the collection of real points $R$. As chaff points aim to confuse an attacker, the cardinality of $C$ is of a greater order of magnitude than the cardinality of set $R$. We define this set of chaff points as illustrated in Equation 5:

$$C = \left\{x' \neq x, y \neq P(x')\right\}, \forall x \in A \ \ \& \ \ \forall x \notin A \ \ \& \ \ |C| >> |R| \tag{5}$$

At this point, we will make this process more clear through Example 3.1, with its illustration in Figure 1.

*Example 3.1.* Let as assume that the secret which Alice wants to lock is $S = 2317$ and that she holds a key, which is the set $K^A = \{1, 2, 3, 4, 5, 6\}$. She creates a polynomial $P$ of degree $n = 3$ which has the secret embedded into its coefficients: $P(x) = 2 * x^3 + 3 * x^2 + 1 * x^1 + 7 * x^0$. The next step involves the projection of her key on the polynomial $P$. Assume that Alice's key is the set $A = \{1, 2, 3, 4, 5, 6\}$ . By projecting each of these points Alice ends up with a collection $R$ of points in a two-dimensional space, $R = \{x, P(x)\}, \forall x \in A$. For our example, this results into: $R = (1, 13), (2, 37), (3, 91), (4, 187), (5, 337), (6, 553)$. In our example, $K^A$'s cardinality is $|K^A| = 6$.

Next, Alice creates the distraction set $C$ comprising of random chaff points which do not lie on the polynomial. Finally, Alice combines and shuffles the two collections in order to create the vault $V$, which contains both genuine and chaff points, which are indistinguishable. As stated before, the size of $C$ should significantly exceed that of $R$. However, for presentation purposes and without harm to the general case, we will only consider four points in the chaff set: $C = \{7, 8, 9, 10\}$ For our toy example, this will yield: $V = \{(1, 13), (2, 37), (3, 91), (4, 187), (5, 337), (6, 553), \ldots, (9, 560), (10, 700)\}$.

**Table 1: Table of symbols used.**

| Symbol | Description |
|---|---|
| n | Degree of polynomial |
| K | Key |
| f | Size of fragment |
| P | Polynomial |
| r | Plaintext record |
| V | Fuzzy Vault |
| RS | Reference Set |
| C | Chaff Points |
| R | Original Points |
| H | Verification Hash |
| Superscripted A ($^A$) | Alice (e.g. $K^A$: Alice's Key) |
| Superscripted B ($^B$) | Bob (e.g. $K^A$: Bob's Key) |
| $\lvert\cdot\rvert$ | Size (e.g $\lvert$RS$\rvert$: Reference Set size) |

*3.3.2 Unlock phase.* Having seen how Alice locks a secret, let us now describe the *Unlock* phase, where Bob will attempt to unlock the Fuzzy Vault $V$ he received from Alice and get access to Alice's secret $S$. For this purpose, Bob will have to use a key set $K^B$ in order to unlock $V$. This will only happen in the case that his key, $K^B$, substantially overlaps with Alice's key $K^A$, used to create $V$. If so, Bob may distinguish enough genuine points of the vault $V$ belonging to the collection $R$.

However, Bob may also identify some distraction points belonging to the chaff set $C$ too. Bob's ability to eventually reconstruct the original polynomial depends on the cardinality of Bob's key. If the key's cardinality is smaller than the polynomial's degree then interpolation methods will not be able to recover the proper polynomial [7]. Therefore, Equation 4 has to be satisfied for Bob's key, $K^B$, as well. Unlocking the vault requires, first, that Bob finds all the vault's points that have the same abscissas with any of his key's elements, or more formally:

$$M = \{(x, y) \in V | x \in K^B\} \tag{6}$$

Distraction points might exist because $K^B$ might also overlap with the collection of random chaff points $C$. If $K^A$ and $K^B$ do not significantly overlap, then the polynomial reconstruction cannot be performed. Example 3.2 illustrates this process.

*Example 3.2.* Let us consider, in our running example, that Bob, with his key set $K^B = \{1, 2, 4, 5, 8, 9\}$, tries to unlock $V$. The collection of $M$ for Bob will be: $M = \{(1, 13), (2, 37), (4, 187), (5, 337), (8, 360)\}$. Bob has to reconstruct $P$ in order to access the vault's contents. For our case, let us consider that this occurs through

**Algorithm 1:** Protocol Overview.

```
   /* Operations at Alice                                    */
1  Build_Fuzzy_Vaults ();
2  Transmit_To_Bob ();
   /* Operations at Bob                                      */
3  Generate_Keys ();
4  Unlock_Alice_Vaults () ; // Privacy Preserving Matching
5  Transmit_Results_To_Alice ();
   /* Operations by both Alice and Bob                       */
6  Transmit_Matching_Records ();
```

Lagrange interpolation [24]. We note here that, part of $K$ for Bob's key is a point $(8, 360)$ which comprises a noise element for $R$ and as such it does not belong to the original polynomial. Consequently, each of the combinations that include the noise element will return a false polynomial. Note here that the cardinality of Bob's key is $|K^B| = |K^A| = 6$. In this case, we have chosen both Alice and Bob to have keys of the same size.

Nevertheless, there is the capacity for Bob to have a key with a different size to Alice's, given that it has at least one element more than the degree of the polynomial used. Since Lagrange interpolation requires $n+1$ genuine points in order to reconstruct the polynomial which corresponds to them, if $|M| < n+1$, Bob will fail to unlock the vault. On the other hand, if $|M| > n+1$ Bob has to create all possible $n+1$ combinations of $M$ and use them in order to interpolate the polynomial. Now, we should take into consideration that each, one of the possible combinations will return a polynomial but Bob has somehow to decide which one is the correct (there is no need to calculate all of them).

### 3.4 Reference Set

A *Reference Set* is a corpus of data used as a means for preprocessing when matching sensitive information. This corpus may either be publicly available or pre-agreed among the matching parties, Alice and Bob, in our case. It works as follows. Considering that Alice and Bob wish to match their data without revealing any information to each other, they use some mapping function, having also been preagreed and the same for both, which relates each of the values in their dataset to one or more values in the Reference Set. Then, instead of comparing their data directly, they are able to compare the results of the mapping function. Such a construct has been used in the privacy preserving record linkage context either for matching [19] or for blocking [13]. Example 3.3 illustrates how a Reference Set may be used.

*Example 3.3.* In this example we are using Levenshtein distance [17], indicated by $L(\cdot)$, as the mapping function. We still consider Alice and Bob, with Alice holding the word 'error', while Bob holds the word 'erasure' and they wish to privately match them. For the sake of presentation, the Reference Set contains the single word 'energy'. In this case, for Alice $L_A(error, energy) = 4$, while for Bob, $L_B(erasure, energy) = 6$. Since $L_A \neq L_B$, we may easily deduce that Alice and Bob hold different values.

## 4 METHOD DESCRIPTION

Having examined the operation of the Fuzzy Vault for numerical data, let us now describe our approach for exploiting this mechanism for approximate private string matching, which will allow us to perform privacy preserving record linkage. We consider two honest but curious matching parties, Alice and Bob. This means that, both of these two participants do not exhibit a malicious behavior, attacking the operation of the protocol, but they try to infer as much information as possible without deviating from the protocol.

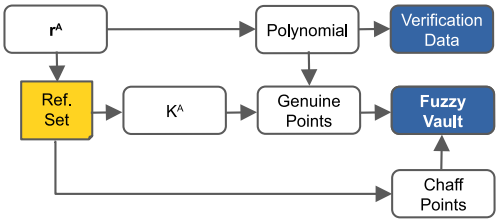



**Figure 2: Steps for locking record $r^A$ into a Fuzzy Vault.**

### 4.1 Protocol Description

Let us begin by providing a simple protocol for using Fuzzy Vaults for privacy preserving record linkage. For this purpose, a separate Fuzzy Vault for each record in a database, as such, a separate polynomial, has to be created and secretly shared. We will consider for our case, that alphanumeric identifiers are used. As our focus is on privately matching records, we assume that either both Bob and Alice share a common schema, or they have different schemas but corresponding fields on the use of which they have agreed beforehand, or they use some privacy-preserving schema matching method.

The protocol we are describing is asymmetric. This means that both parties do not perform exactly the same steps, but they cooperate in order to achieve their common goal of privately linking their records. These steps, are illustrated in Algorithm 1. First, Alice converts all of the records in her database to Fuzzy Vaults, resulting in one Fuzzy Vault for each record. Then, she transmits these Fuzzy Vaults to Bob through a considered secure channel. Bob, in his turn, creates his Fuzzy Vaults and, using the resulting keys, attempts to unlock the received vaults, resulting in a list of matches. When the process concludes, he sends the ids of the matching Fuzzy Vaults back to Alice. The procedure concludes with both Alice and Bob exchanging their actual matching records.

At this point, the question that rises is how these alphanumeric fields are going to be transformed into polynomials and, eventually, into Fuzzy Vaults. This is the process we are going to describe next.

### 4.2 String Encoding for Locking

Beginning with the locking phase, Alice will secure strings inside Fuzzy Vaults. Each string is the concatenation of all the string fields in the record that will be used for matching. In order for such a string to be locked, and to be able to be unlocked afterward by Bob by a fairly similar string, it should be somehow associated with the polynomial used to lock it. As such, our method embeds irreversibly the string into polynomial's coefficients. We cannot use a direct representation of the string since this would jeopardize exposing the key as the abscissas of the genuine points of the vault, allowing an adversary obtaining the key to directly recover the string. Additionally, as we will describe later on, Bob will use this property after unlocking the Fuzzy Vaults, in order to verify if the match he achieved with Alice's secret is identical or approximate.

To be able to adapt the Fuzzy Vault scheme so as to exhibit privacy preservation properties for record linkage, a series of

**Algorithm 2:** Mapping a Record to a Polynomial.

**Input:**
- **r:** A record with alphanumeric fields
- **f:** Size of fragment
- **n:** Size of polynomial

**Output:**
- **p:** A polynomial representation of $r$

```
1  ncount ← 0;
2  s ← Concatenate (r);
3  fragments[] ← Fragment(f, s);
4  foreach fragment ∈ fragments[] do
5  |   ASCII_Codes ← ∅;
6  |   foreach char ∈ fragment do
7  |   |   ASCII_Codes.append (to_ASCII (char));
8  |   end
9  |   SetCoefficient (ASCII_Codes, n − ncount);
10 |   ncount+ = 1;
11 end
12 for i ← ncount to n by 1 do
13 |   SetCoefficient (1, n − i);
14 end
```

steps is necessary. The broad image of these steps are illustrated in Figure 2. First of all, a record, which plays the role of the string to be secured, plays a double role in the whole process, as it is used both for creating a polynomial and for building a key for encoding this record. This polynomial and the key are then used to create the set of Genuine Points, which, in combination with the generated chaff points, will result in a Fuzzy Vault. At this point, we have to mention that the polynomial representation of the record is also used to produce the verification data, also used during the unlocking phase.

*4.2.1 Polynomial generation.* Let us begin with examining the process that generates the aforementioned polynomial. The mapping algorithm we propose is illustrated in Algorithm 2. First, the alphanumeric fields of the record to be locked are concatenated so as to form a single string (line 2). Then, this string is fragmented into substrings of size equal to $f$ (line 3), with $f$ being a user defined parameter, an approach we adopted for security purposes. This is particularly useful for long strings. Each character of each fragment is converted to its ASCII code equivalent. The concatenation of these codes creates a number, which will form a polynomial's coefficient (lines 4-11). The remaining coefficients of the polynomial are set to 1. Example 4.1 further clarifies the operation of this algorithm. On the other hand, if the string exceeds the length of 27 characters either a higher degree polynomial should be created or some characters should be skipped.

*Example 4.1.* Let us assume, as illustrated in Figure 3, that $f$ = 3, meaning that the fragment size will consist of three characters and that we wish to encode the name *JOE_DOE*. Let us also assume that the degree of the polynomial we wish to use is $n$ = 8. This will yield a polynomial of the form: $P(x) = c_8 \cdot x^8 + c_7 \cdot x^7 + \ldots + c_2 \cdot x^2 + c_1 \cdot x^1 + c_0$. Given our settings for $n$ and $f$, this polynomial has a capacity of embedding $9 \cdot 3 = 27$ characters, because it has 9 coefficients. Moreover, there are $26^3 = 17576$ possible combinations for each coefficient, because each coefficient can have up to three letters and there are 26 available letters (we take into consideration only capital letters) and $26^{3*9}$ total possible

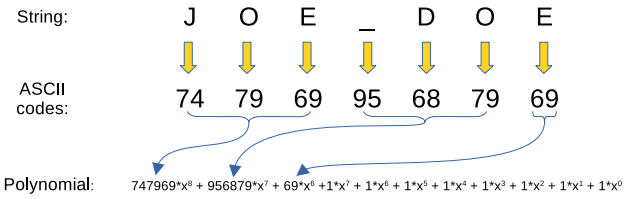



**Figure 3: Example of polynomial representation for the concatenated name and surname "JOE_DOE".**

polynomials. In our example, the encoded string has less than 27 characters. In this case, the remaining coefficients are all set to 1.

*4.2.2 Verification.* The aim of the verification step is, as its name suggests, to verify at the unlocking phase that the recovered polynomial matches the encoded one. In our approach, we use the method proposed in [20]. This method, is based on auxiliary data generated during the polynomial construction phase. An MD5 hash $H$ which derives directly from the created polynomial, is created having as input the concatenation of the polynomial's coefficients.

*4.2.3 Key Generation.* Building a polynomial is one of the two steps required toward generating a record's set of genuine points. The second step regards the generation of the key. Since the locking and the unlocking keys are the two components which are being compared, the key in our method should be directly linked with the string. The caveat here is to avoid an one - to - one connection between the key and the string as a measure against record multiplicity attacks, described in [23]. For instance, in the case that an adversary somehow gets access to the key, our method should not allow the recovering of the secret string.

As such, the need to devise an one-way mapping emerged. To this end, we employ a Reference Set in order to associate its values with the string resulting from the record's concatenated fields by means of a string similarity function. Then, the most similar values are selected and they are transformed, in an irreversible manner, to a set of numbers comprising the key for the locked string, without, however, revealing any information about it.

Let us now see in how this works. The generation of a locking key for a single record is detailed in Algorithm 3. This algorithm should be applied on each record in the recordset resulting in a set of separate keys, one for each record. This algorithm requests as input the concatenated alphanumeric fields of record $r$, as a single string, to be encoded, a Reference Set $RS$ that we are going to use for encoding and the size $|K|$ of the key $K$ which we are going to produce.

Our algorithm also requires the use of a string similarity, or distance, function $F(\cdot)$. This is applied on the pairs formed by $r$ and each of the values of the Reference Set $RS$, $RS\_value$ resulting to an array of similarity scores $sim\_scores$ (lines 1-4). $sim\_scores$ is then used to partially sort $RS$. As our aim is to retrieve the to $|K|$ values of the $RS$, a full sort is unnecessary. The sort order depends on the type of measure used for $F(\cdot)$. If $F(\cdot)$ is a similarity function, then sorting occurs in descending order, as we need to retrieve the most similar of $RS$'s values with $r$. Otherwise, sort in ascending order is performed. Afterward, $top\_k\_values$ are extracted from $RS$, which are the top $|K|$ most similar Reference Set values with $r$ (lines 6-9). Next, we are going to build $K$ by applying a series of transformations on each of $top\_k\_values$, resulting in a numerical set.

**Algorithm 3:** Key Generation.

**Input:**
- **r:** A record with concatenated alphanumeric fields
- **RS:** Reference Set
- **|K|:** Size of resulting key

**Output:**
- **K:** Resulting key

```
1  sim_scores ← ∅;
2  foreach RS_value ∈ RS do
3  |  scores.append (F (r, RS_value));
4  end
5  Partial_Sort (RS, sim_scores, |K|);
6  top_k_values ← ∅;
7  for i ← 0 to |K| by 1 do
8  |  top_k_values.append (RS[i]);
9  end
10 K ← ∅;
11 for i ← 0 to |K| by 1 do
12 |  hash ← Hash (top_k_values[i]);
13 |  seed ← hash;
14 |  random_integer ← Rand (seed, 0, 10^6);
15 |  K.append (Cosine (random_integer));
16 end
```

The transformations that each string $top\_k\_values[i]$ undergoes are as follows (lines 10-16). Initially, $top\_k\_values[i]$ is hashed through a simple hash function so as to create a numerical equivalent *hash* (line 12). Then, *hash* is used as an input seed in a random number generator which produces *random_integer* ranging between $[0, 10^6]$ (lines 13-14). The series of transformation conclude with applying the cosine function on the *random_integer*, resulting in a value between $[-1, 1]$, which is now a part of $K$ (line 15).

It is evident that after applying this series of mappings we have managed to perform an one-way transformation between the initial string that consists of the concatenation of the records fields and the numbers that comprise $K$. This is due to the use of the numerical representation of the string as a seed and the periodic nature of the cosine function.

*4.2.4 Fuzzy Vault Construction.* At this point, three steps remaining for constructing a Fuzzy Vault: Generating Genuine Points, generating Chaff Points and, finally, blending these together.

*Genuine Points Generation.* Now that we have created the key $K$ and the polynomial $P$ required by the Fuzzy Vault scheme to encode a secret, we will projected $K$ over $P$, as indicated in [12]: $R = \{P(x)\}, \forall x \in K$ resulting in a set of $K$ points considered to be the set $R$ of genuine points in the Fuzzy Vault.

*Noise as Chaff Points.* After the creation of the genuine points, chaff points should also be generated ensuring fuzzy vault's security. Noise comprises a standard method for enhancing privacy [4]. In our method we adhere to this principle by generating chaff points which aim at distracting a potential attacker from identifying the points that belong to the actual polynomial. A naive approach would be to randomly generate points based on one or more distributions. However, we consider that this would allow for identifying and phasing out noise quite easily. As such, we designed a novel approach, more suitable for our case, which is based on the use of Reference Sets. We use abscissas from a Reference Set we employ, while the ordinates are randomly chosen but with the restriction that the resulting points do not lie on the polynomial. This way, $C$ is built.

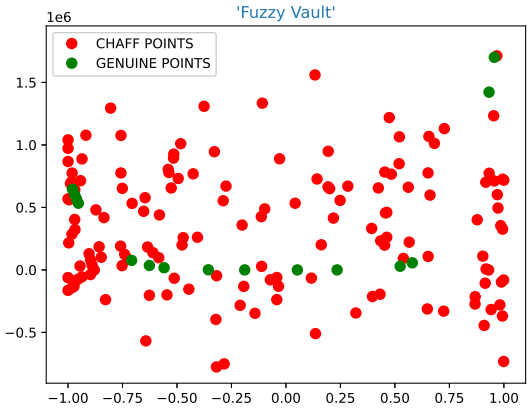



**Figure 4: Fuzzy vault where chaff points have abscissas from the Reference Set and random ordinates.**

*Final Step.* Eventually, the points of these sets are shuffled in order for genuine and chaff points to be mixed, applying formula 5 so as to create $V$. A Fuzzy Vault created with this method is illustrated in Figure 4. At this point, the data encoding from Alice's side concludes. Alice sends Bob the Fuzzy Vaults she has constructed together with the verification hash, One pair for each record. At this point the first step (line 1) of the protocol illustrated in Algorithm 1. To further clarify our approach, we will use Example 4.2:

*Example 4.2.* Let us assume that "JOHN_SNOW" is the string Alice wishes to lock. Let us also assume that $F(\cdot)$ is going to be Levenshtein distance [17]. First, she encodes the string into a polynomial, using the fragmentation of the respective ASCII codes, with $f = 3$: $P(x) = 747972x^8 + 789583x^7 + 787987x^6 + 1x^5 + 1x^4 + 1x^3 + 1x^2 + 1x^1 + 1x^0$.

Based on the representation of the polynomial, she also constructs the verification hash, by concatenating the coefficients of the polynomial so as to form "747972786583787987111111". Applying MD5 results in: "2d73f7e7425e2d20264a799e24715317", which will be delivered together with the Fuzzy Vault $V$ to Bob.

Then, Alice will have to build her key $K^A$. For this purpose, she selects the top $|K^A| = 15$ elements of the Reference Set which are more similar with the word "JOHN_SNOW". For instance, the first element is "JONES_OWEN" which has F(JOHN_SNOW, JONES_OWEN) = 5, while the last one is the string "XU_BO" with F(JOHN_SNOW, XU_BO) = 7. Now, we hash each of the strings in $K^A$ into numbers and then the numbers are mapped into the range $[-1, 1]$ using the cosine function: $COS(JONES\_OWEN) = -0.355866$.

Next, in order to create the genuine points, Alice projects the elements of her key $K^A$ onto the polynomial $P$. For instance, for the first key element this yields $P(-0.355866) = 1222.8648$. The rest are accordingly projected. When this step concludes, $R$ will have been built.

Eventually, Alice will have to build the set of chaff points $C$ and then calculate its union with $R$ so as to build the Fuzzy Vault $V$. $V$ coupled with the verification hash $H$ are delivered to Bob.

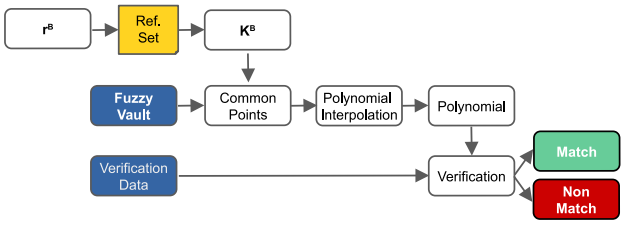



**Figure 5: Steps for linking record $r^B$ with a secured record inside a Fuzzy Vault.**

## 4.3 Matching through Unlocking

At this point, we assume that the second step (line 2) of Algorithm 1 has been successfully completed, thus Alice's records have been successfully transmitted to Bob in the form of Fuzzy Vaults, accompanied by their Verification Data and we are moving to steps 3 and 4. Now, Bob will attempt to unlock these Fuzzy Vaults, thus, linking his records with Alice's. To do so, he will have to perform the actions illustrated in Figure 5. First, he will have to create a key for each of his records. Then, all of his keys are going to be tested against each of Alice's Fuzzy Vaults, also using each vault's Verification Data. Let us see these in detail.

*4.3.1 Verification.* As described in Section 3.3, a Fuzzy Vault is unlocked by a key similar to the one that locked it. It is evident that, for this to happen, Bob should create his unlocking keys in way identical to the process that Alice followed to create her locking keys. Thus, Bob builds each of his unlocking keys $K^B$, for each of his records, using Algorithm 3. Afterwards, he will calculate the intersections of all the unlocking keys with all the Fuzzy Vaults, forming a series of *Common Sets*, each of which contains $(x, y)$ pairs. Each of Common Set $M$ is then used for reconstructing the polynomial. Adhering to the literature [6, 20, 25], this achieved through Lagrange interpolation [7] on the common points $M$, to create an interpolating polynomial.

## 4.4 Approximate Matching

At this point, our methodology for approximate matching is applied. The key point for achieving approximation with a Fuzzy Vault is the step of the reconstruction of the polynomial described by the Fuzzy Vault. As mentioned earlier, an interpolated polynomial of $n$-th degree needs at least $n + 1$ interpolating points in order to be created. In our case, the polynomial is secured in a Fuzzy Vault as $|K^A|$ genuine pairs $(x, y)$. Therefore, when $|A| = n + 1$, in order to interpolate an $n$-th degree polynomial, the two keys have to be identical.

However, if we build a key $K^A$, having a size greater than the minimum number of points that an $n$-th degree polynomial needs in order to be recovered, i.e., $|K^A| > n + 1$ we can exploit these redundant $(x, y)$ pairs to achieve approximation. In other words, since $n + 1$ points are required to unlock an $n$-th degree polynomial and we project a key with size larger than $n + 1$, we may use any $n + 1$ points of $K^A$ to unlock the Fuzzy Vault. As such, $K^B$ should only match $n+1$ of $K^A$'s points, with $|K^A| > n+1$, thus leading to linking non-identical records. On the other hand, as $|K^A|$ increases, the probability of having common elements in different keys is increased too. As such, too large keys would result into increased numbers of fault positives.

Now, after the recovering the polynomial from $V$, Bob, creates a Verification Hash $H^B$ in an identical way that Alice created $H^A$. Then these two are checked for matching. If $K^B$ sufficiently overlaps $K^A$ and $H^A = H^B$, the two polynomials will be the same and there is a match. On the other hand, if the two sets are not similar enough, verification will fail. If the polynomial is classified as a match, then the strings for which the vault has been created is considered to be the same with the string Bob holds. To facilitate the reader, we now provide Example 4.3:

*Example 4.3.* Continuing Example 4.2 Bob tries to unlock a Fuzzy Vault he received from Alice containing the record "JOHNY_SNOW". First, he creates his key the same way Alice did, using the same fragment size. After the creation of his key, Bob has to find the common points between his key and the fuzzy vault. In this case, Bob managed to find n = 10 common points. Thereby, by utilizing Lagrange interpolation, he managed to recover a polynomial of 8th degree by using 9 out of the 10 common points (he might have to try all the combinations) as shown: $P_B(x) = 747972x^8 + 789583x^7 + 787987x^6 + 1x^5 + 1x^4 + 1x^3 + 1x^2 + 1x^1 + 1x^0$. Last but not least, Bob has to verify the recovered polynomial, because he has no knowledge if it is the correct one or not. Therefore, he calculates the MD5 hash of the concatenation string of his polynomial's coefficients, which in our case is "2d73f7e7425e2d20264a799e24715317". Comparing the two hashes Bob can asses that the two strings match.

## 4.5 Protocol Conclusion

Eventually, after Bob has decided which of his records match Alice's, he transmits a list of the matching records he recovered requesting their entire data. Alice responds and she exchanges her corresponding data with Bob (lines 5, 6 of Algorithm 1).

# 5 PRIVACY DISCUSSION

In our setup, we consider that Alice and Bob exhibit an honest but curious behavior, meaning that they try to gather as much additional information they can by adhering to the protocol.

A Fuzzy Vault's security properties rely on the number of chaff points used while constructing it [12]. We remind our notation, where $n$ represents the degree of the polynomial used, $|C|$ the number of generated chaff points and $|R|$ the number of real, genuine points. Taking into account that, in order to have a correctly reconstructed polynomial, $n + 1$ of the genuine points must be used. Therefore, any adversary who has access to a fuzzy vault and wants to unlock it has to find all possible $n + 1$ combinations of points and make reconstruction attempts. Even in this case the adversary has to know the polynomial degree $n$ which has been used.

Without any extra information (rather than the polynomial degree) any adversary is required to find $\binom{|R|+|C|}{n+1}$ point combinations. From all the combinations of $n + 1$ points that can be extracted by the vault only $\binom{|R|}{n+1}$ combinations are able to unlock it. Therefore, the probability of guessing correctly a needed combination is given by Equation 7.

$$\frac{\binom{|R|}{n+1}}{\binom{|R|+|C|}{n+1}} \tag{7}$$

Consequently, a brute force attack in a well concealed fuzzy vault is considered impracticable. In our approach, to further increase privacy, we employ a series of further enhancements. First, there are the chaff points, which comprise the Fuzzy Vault. Then, there is the use of a Reference Set and, finally, the use of non-bijective mapping functions.

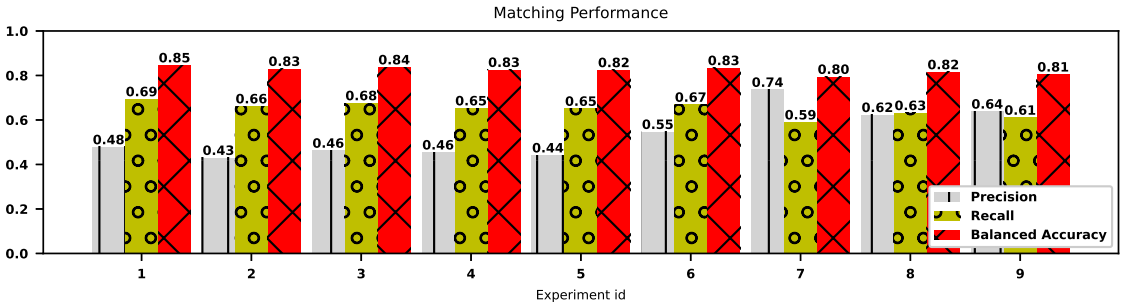



**Figure 6: Fuzzy Vault Matching Performance.**

Chaff points are responsible for concealing the vault's secret which, in our case, is a record . We have taken the following measures in order to further enhance this property. First of all, regarding the use of the Reference Set and mapping functions. Sensitive data are not encoded themselves within the Fuzzy Vault. What is encoded with the Reference Set is the output of a distance or similarity function. Furthermore, the output of this function is fed to a non-bijective function, thus constituting both the polynomial representation and the key generation processes irreversible.

To continue, to avoid frequency attacks, instead of using a random function to generate points, the chaff generation technique we propose is based on the use of a Reference Set. This way, the abscissas of the points delivered to Bob are indistinguishable from the original points. Consider that the vault has 15 genuine points and 200 chaff points which their abscissas os also derived from the Reference Set, the same with the genuine points. Bob knows all the possible abscissas which are produced by the Reference Set but our method uses these abscissas as noise. Therefore, there is no information leakage which can help Bob to unlock the vault without having a proper key.

Furthermore, At this point, it should be noted that, in contrast with the biometrics applications utilizing Fuzzy Vaults, [6, 25] the leakage of a key may only affect a single vault. This is due to the fact that we employ a separate polynomial for each string to be locked in the Fuzzy Vault. Additionally, the degree of the polynomial and the size of the chaff set also affect a Fuzzy Vault's offered privacy. For further hardening the Fuzzy Vault, the degree of the polynomial may be increased. Increasing the size of the chaff set will as well favor privacy.

## 6 EXPERIMENTAL EVALUATION

In this section, we will provide empirical evidence regarding the matching performance and the time characteristics of our method. Finally, we will compare the Fuzzy Vault method we propose against a baseline privacy preserving matching method based on phonetic codes [14]. For our evaluation, we have implemented a prototype in Python 3[1] enhanced by a series of packages[2].

### 6.1 Setup

In our experiments, we have used samples from real-world data originating from the North Carolina voters database[3]. To evaluate our approach, we have used 'last name' and 'first name' from this database as matching attributes, assuming they comprise a candidate key. Sampling this database, we constructed two datasets, dataset_A and dataset_B. Dataset_A initially consisted of 1000 records and then deduplicated to 990 ones. Dataset_B has been built from dataset_A. Since our method is capable of achieving approximate matching and in order to evaluate its performance in this terms, we corrupted Dataset_B so as to contain one error per attribute, as it is rather unusual for a word to contain more than one to two typographical errors [8]. Experiments were conducted on an 8-core virtual server with 16GB of RAM, utilizing, however, only a single core was for the assessment each time.

For the Reference Set, now, there are two aspects we are interested in. Its content and its size. As for the content, we used names and surnames originating from the initial database. In terms of Reference Set sizes, we experimented with three distinct dataset sizes. For each letter of the latin alphabet, 15, 20 and 25 records were selected with a uniform distribution, resulting in $15 * 26$ =390 records, $20 * 26 = 520$ records and $25 * 26 = 650$ records. For each size we created three different Reference Sets in order to have more information about the effect of the set's content on the results. Table 2 contains the configurations for all these Reference Sets. Since for each size of Reference Set we will use three different Reference Sets, each time we perform an experiment with each of them, we maintain a fixed id. To this end, experiment ids 1–3 are for the Reference sets of 390 elements, 4–6, for those of 520 ones and 7–9 for those with 600 elements.

The evaluation of our method is going to be based on metrics broadly used in information retrieval and data mining. These metrics are Precision, Recall and Balanced Accuracy. Precision is the fraction of the relevant elements among the retrieved elements ($Precision = \frac{\text{TP}}{TP+FP}$). Recall is the fraction of the retrieved elements divided by the total relevant elements ($Recall = \frac{\text{TP}}{TP+FN}$). Balanced Accuracy is used as a metric to indicate accuracy in cases of data with imbalanced classes, where the number of instances of one class, overwhelms is significantly larger than the number of instances of the other one. This metric normalizes the true positive and true negative predictions by the number of actual positive and negative samples, respectively, and divides their

**Table 2: Reference sets used in the evaluation.**

| Reference Set id | Samples per letter | Size |
|---|---|---|
| 1, 2, 3 | 15 | 390 |
| 4, 5, 6 | 20 | 520 |
| 7, 8, 9 | 25 | 650 |

[1]https://github.com/XhinoMullaymeri/Fuzzy_Vault_PPRL_DOLAP21
[2]Python 3 packages: hashlib, random, scipy.interpolate.lagrange, numpy, numpy.polyfit, Levenshtein
[3]Available at: https://dl.ncsbe.gov/index.html?prefix=data/

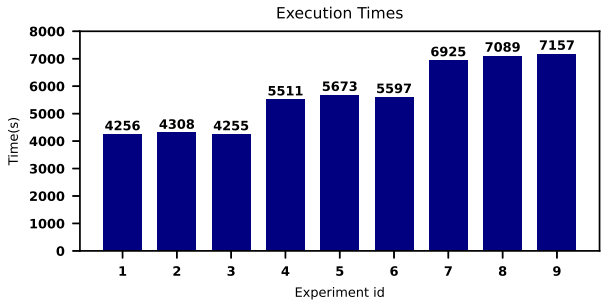



**Figure 7: Fuzzy Vault Execution Time.**

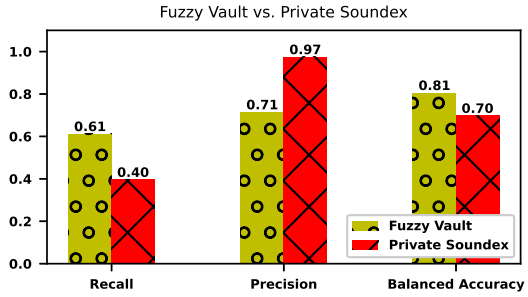



**Figure 8: Fuzzy Vault vs. Privacy Preserving Soundex**

sum by two as shown : $BA = \frac{TPR+TNR}{2}$, where $TPR = \frac{\text{TP}}{TP+FN}$ and $TNR = \frac{\text{TN}}{TN+FP}$.

In our experiments, we chose $n = 8$ for the polynomials we used and the key sizes to be equal to 15. Our interpolation method was Lagrange interpolation. Also, we have fixed the number of chaff points to 150, which were selected from the Reference Set using a Normal distribution. Concerning the algorithm for comparing the Reference Set with the strings resulting from the records of our dataset we used Levenshtein distance [17].

## 6.2 Experimental results

Next, we will examine in detail the outcomes of our empirical evaluation.

*6.2.1 Matching Performance.* Let us begin our experimental assessment by examining the matching performance achieved by our Fuzzy Vault scheme. This is illustrated in Figure 6. The horizontal axis represents the ids of the Reference Sets used, while the vertical one represents the value for the respective metric, all ranging between 0 and 1. Regarding the results, now, first of all, we may observe that Recall levels, represented by the yellow bars, are almost for every test above 60%. This means that at least 60% of the fuzzy vaults which had been created using dataset_A, managed to be unlocked by the proper record of dataset_B resulting in a successful record linkage. Furthermore, we may observe three clusters forming, one for each Reference Set size, where the results are similar within each cluster.

Comparing precision and recall, we may observe that they follow opposite directions, a fact that constitutes an expected outcome. Related with the Reference Set used, the following situation occurs. When smaller Reference Sets are employed, more keys are assigned to the same item of the Reference Set. Thus, more keys are associated. On the other hand, larger Reference Sets associate small numbers of keys in each of their elements, leading to higher precision, yet lower recall.

It is evident that the size of the Reference Set used plays an important role in the overall behavior of our method. Nevertheless, we have to stress that no particular measures were taken to optimize the outcome regarding the distribution of the values of the Reference Sets. We have deliberately chosen this route in order to illustrate the baseline characteristics our approach exhibits.

Considering Balanced Accuracy now, it is easy to see that it remains at very high levels in all cases. This is an interesting result by itself, since it is not only for the matching pairs of records we are interested, but also, for the non-matching ones. Considering that we aim at an 1-1 matching for each of dataset_A's records with the ones of dataset_B. Considering 1,000 records for each dataset, we would ideally end up with 1,000 pairs matched and $1,000 * 1,000 - 1,000 = 999,000$ pairs not matched. Precision and recall do not take into account such imbalances, a situation captured, however by Balanced Accuracy.

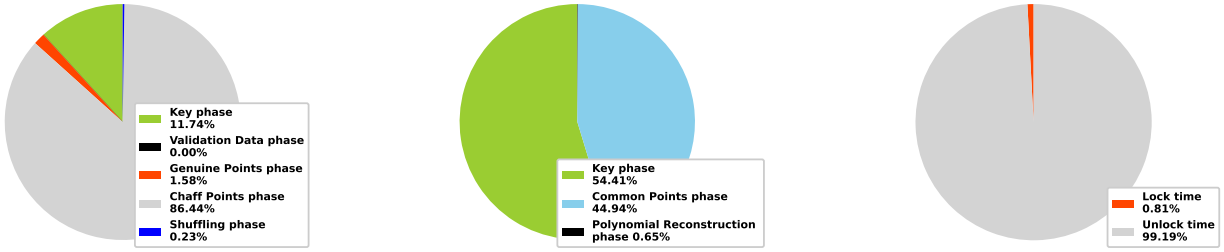
*6.2.2 Total Execution Time.* In this set of experiments, we will assess the relation between the execution time and the size of the Reference Set. The results of this evaluation are illustrated in Figure 7. The vertical axis represents time in seconds, while the horizontal one, again, stands for the Reference Set used. The first observation we can make is that there are groups formed, based on the size of the Reference Set. Within each group, execution times are similar, as the sizes of these Reference Sets are identical and for each key the same number of comparisons has to take place. On the other hand, as the size of the Reference Set increases, execution times increase as well. As a result, we can deduce that, total execution time is proportional to the size of the Reference Set.

This observation is particularly useful, especially when combined with the conclusions from the previous set of experiments. In more detail, using smaller Reference Sets has the result of higher recall and lower precision. On the other hand larger Reference Sets increase, not only execution time but also precision, with the cost of lower recall. Selecting intermediate values for the Reference Sets offer a fair trade-off between precision and recall and average time performance.

*6.2.3 Locking and unlocking time shares.* In this experiment, we will focus on Reference Set 8, in order to illustrate time shares for locking and unlocking the vaults. These are illustrated in Figures 9a and 9b. We will begin by describing the locking phase. Here as we may see, the time required by the locking phase is mostly occupied by the chaff points generation. This is, however, one of the basic elements for offering strong privacy characteristics. The rest of the steps of the locking step occupy only a small portion of the overall time required, making this part of our approach open for further improvements.

Regarding unlocking, it is easy to see that for unlocking, time is almost equally shared between reconstruction and key generation. Identifying common points requires just a tiny portion of the overall time.

At this point, we will make use of one more plot to be able to extract safe conclusions regarding execution time. As we may see, Figure 9c illustrates a time comparison between the locking and the unlocking phase. It is evident, that the time required to unlock all vaults is much larger than the time required to lock them. In particular, while 58 seconds are required for encoding, 6869 seconds are required for both decoding and performing an




**(a) Locking.** **(b) Unlocking.** **(c) Successful Unlocking.**

**Figure 9: Execution time shares.**

all-to-all comparison, since exhaustive all-to-all matching was employed in order to allow us extracting safe conclusions.

*6.2.4 Comparison with privacy preserving Soundex.* Finally, we conclude our experimental evaluation, comparing against Privacy Preserving Soundex [14]. For this experiment, we used Reference Set 7, comprising of 650 entries.

This experiment is illustrated in Figure 8. Again, the vertical axis displays values of the metrics, while the horizontal one represents the name of each metric. Green bars correspond to Privacy Preserving Soundex, while our method is illustrated with the red bars. We may observe that only in terms of Precision our method is inferior to Privacy Preserving Soundex. This, however, is a result of the low Recall that Privacy Preserving Soundex has exhibited. On the other hand, Fuzzy Vault manages to exhibit a more balanced behavior. This result is promising, also taking into account that Privacy Preserving Soundex is a three-party method, as opposed to our two-party method. Regarding execution time, Private Soundex operates in less than a second. However, as the results are simulated and this protocol requires a three-party setting for achieving privacy, we may not extract valid conclusions based on this comparison, since protocol communication times are not taken into account.

## 7 CONCLUSIONS AND FUTURE WORK

In this paper, we presented the details of our method for performing two-party privacy preserving record linkage using Fuzzy Vaults. We discussed the privacy properties of our approach and provided extended empirical evidence that this new approach is very promising. In our next steps, we aim at further refining our method aiming at increasing matching performance while, at the same time, improving execution times, maintaining, nevertheless, our method's privacy characteristics. We will further experiment with tuning our method's parameters, trying alternative interpolation methods.

## REFERENCES


[1] Luca Bonomi, Yingxiang Huang, and Lucila Ohno-Machado. 2020. Privacy challenges and research opportunities for genomic data sharing. *Nature Genetics* (2020), 1–9.

[2] Feng Chen, Xiaoqian Jiang, Shuang Wang, Lisa M Schilling, Daniella Meeker, Toan Ong, Michael E Matheny, Jason N Doctor, Lucila Ohno-Machado, and Jaideep Vaidya. 2018. Perfectly secure and efficient two-party electronic-health-record linkage. *IEEE internet computing* 22, 2 (2018), 32–41.

[3] Yanling Chen. [n. d.]. Current approaches and challenges for the two-party privacy-preserving record linkage (PPRL). *Collaborative Technologies and Data Science in Artificial Intelligence Applications* ([n. d.]).

[4] Peter Christen, Thilina Ranbaduge, and Rainer Schnell. 2020. *Linking Sensitive Data: Methods and Techniques for Practical Privacy-Preserving Information Sharing*. Springer International Publishing AG.

[5] Peter Christen, Thilina Ranbaduge, Dinusha Vatsalan, and Rainer Schnell. 2018. Precise and fast cryptanalysis for Bloom filter based privacy-preserving record linkage. *IEEE Transactions on Knowledge and Data Engineering* 31, 11 (2018), 2164–2177.

[6] T Charles Clancy, Negar Kiyavash, and Dennis J Lin. 2003. Secure smartcardbased fingerprint authentication. In *Proceedings of the 2003 ACM SIGMM workshop on Biometrics methods and applications*. 45–52.

[7] Samuel Daniel Conte and Carl De Boor. 2017. *Elementary numerical analysis: an algorithmic approach*. SIAM. 31–41 pages.

[8] Fred J Damerau. 1964. A technique for computer detection and correction of spelling errors. *Commun. ACM* 7, 3 (1964), 171–176.

[9] Aleksander Essex. 2019. Secure Approximate String Matching for Privacy-Preserving Record Linkage. *IEEE Transactions on Information Forensics and Security* 14, 10 (2019), 2623–2632.

[10] Michael T Goodrich. 2009. The mastermind attack on genomic data. In *2009 30th IEEE Symposium on Security and Privacy*. IEEE, 204–218.

[11] Ali Inan, Murat Kantarcioglu, Gabriel Ghinita, and Elisa Bertino. 2010. Private record matching using differential privacy. In *Proceedings of the 13th International Conference on Extending Database Technology*. 123–134.

[12] Ari Juels and Madhu Sudan. 2006. A fuzzy vault scheme. *Designs, Codes and Cryptography* 38, 2 (2006), 237–257.

[13] Alexandros Karakasidis, Georgia Koloniari, and Vassilios S Verykios. 2015. Scalable blocking for privacy preserving record linkage. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 527–536.

[14] Alexandros Karakasidis and Vassilios S Verykios. 2009. Privacy preserving record linkage using phonetic codes. In *2009 Fourth Balkan Conference in Informatics*. IEEE, 101–106.

[15] Basit Khurram and Florian Kerschbaum. 2020. SFour: A Protocol for Cryptographically Secure Record Linkage at Scale. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 277–288.

[16] Mehmet Kuzu, Murat Kantarcioglu, Ali Inan, Elisa Bertino, Elizabeth Durham, and Bradley Malin. 2013. Efficient privacy-aware record integration. In *Proceedings of the 16th International Conference on Extending Database Technology*. 167–178.

[17] Vladimir Levenshtein. 1965. Binary codes capable of correcting spurious insertions and deletion of ones. *Problems of information Transmission* 1, 1 (1965), 8–17.

[18] Xhino Mullaymeri and Alexandros Karakasidis. 2021. A Two-Party Private String Matching Fuzzy Vault Scheme. In *The 36th ACM/SIGAPP Symposium On Applied Computing*. ACM.

[19] Chaoyi Pang, Lifang Gu, David Hansen, and Anthony Maeder. 2009. Privacy-preserving fuzzy matching using a public reference table. In *Intelligent Patient Management*. Springer, 71–89.

[20] Wendy Ponce-Hernandez, Ramon Blanco-Gonzalo, Judith Liu-Jimenez, and Raul Sanchez-Reillo. 2020. Fuzzy Vault Scheme Based on Fixed-Length Templates Applied to Dynamic Signature Verification. *IEEE Access* 8 (2020), 11152–11164.

[21] Thilina Ranbaduge, Peter Christen, and Rainer Schnell. 2020. Secure and Accurate Two-Step Hash Encoding for Privacy-Preserving Record Linkage. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 139–151.

[22] Ahsan Saleem, Abid Khan, Furqan Shahid, M Masoom Alam, and Muhammad Khurram Khan. 2018. Recent advancements in garbled computing: how far have we come towards achieving secure, efficient and reusable garbled circuits. *Journal of Network and Computer Applications* 108 (2018), 1–19.

[23] Walter J Scheirer and Terrance E Boult. 2007. Cracking fuzzy vaults and biometric encryption. In *2007 Biometrics Symposium*. IEEE, 1–6.

[24] Springer Verlag GmbH, European Mathematical Society. [n. d.]. Encyclopedia of Mathematics. Website. URL: https://www.encyclopediaofmath.org/. Accessed on 2020-08-11.

[25] Umut Uludag, Sharath Pankanti, and Anil K Jain. 2005. Fuzzy vault for fingerprints. In *International Conference on Audio-and Video-Based Biometric Person Authentication*. Springer, 310–319.

[26] Dinusha Vatsalan, Ziad Sehili, Peter Christen, and Erhard Rahm. 2017. Privacy-preserving record linkage for big data: Current approaches and research challenges. In *Handbook of Big Data Technologies*. Springer, 851–895.