Privacy Preserving Record Matching Using Automated Semi-Trusted Broker

Ibrahim Lazrig[1], Tarik Moataz[1,2], Indrajit Ray[1], Indrakshi Ray[1], Toan Ong[3], Michael Kahn[3], Frédéric Cuppens[2], and Nora Cuppens[2]

[1] Dept. of Computer Science, Colorado State University
{lazrig, tmoataz, indrajit, iray}@cs.colostate.edu
[2] Institut Mines-Télécom, Télécom Bretagne, Cesson Sévigné, France
{frederic.cuppens,nora.cuppens}@telecom-bretagne.eu
[3] Anschutz Medical Campus, University of Colorado, Denver
{Toan.Ong, Michael.Kahn}@ucdenver.edu

Abstract. In this paper, we present a novel scheme that allows multiple data publishers that continuously generate new data and periodically update existing data, to share sensitive individual records with multiple data subscribers while protecting the privacy of their clients. An example of such sharing is that of health care providers sharing patients' records with clinical researchers. Traditionally, such sharing is performed by sanitizing personally identifying information from individual records. However, removing identifying information prevents any updates to the source information to be easily propagated to the sanitized records, or sanitized records belonging to the same client to be linked together. We solve this problem by utilizing the services of a third party, which is of very limited capabilities in terms of its abilities to keep a secret, secret, and by encrypting the identification part used to link individual records with different keys. The scheme is based on strong security primitives that do not require shared encryption keys.

1 Introduction

Many applications exist where a group of data sources (publishers) continuously generate sensitive data, periodically update the same, and share the data with another group of data analyzers (subscribers). To protect the privacy of the clients of the publishers, the data sharing needs to occur in a privacy-preserving manner, which in its simplest form is enabled by removing identifying information from the data. An example of such data sharing is observed in the so-called clinical data-sharing networks. Different health care providers (e.g., medical clinics, laboratories, hospitals and pharmacies) are the publishers of the data for the networks while clinical researchers are the subscribers of the data. Unlike the traditional privacy-preserving data publishing domain, the data in such clinical data-sharing networks are not static but are updated every time a patient interacts with a data publisher.

Owing to the updatable nature of the data, a unique and challenging situation occurs in such applications that is not observed in traditional privacy preserved

data publishing setups. Any updates to a record on the publisher side must be pushed to the corresponding record on the subscriber side even though these two records have been delinked via sanitization algorithms. Consider the clinical data-sharing example. Assume that a clinical researcher needs data related to a specific demography. In this case, patients' identification information (such as SSN, driver's license number, date of birth, etc.) are typically removed when the medical information is shared with the researcher. To provide the most relevant and current data, patients' progress under treatment regimens would need to be propagated to the clinical researcher. Similarly, the researcher should also be able to pull updates from the publisher or at a minimum be able to query the publisher for updates. To allow such sharing of information, records at the publisher need to be somehow linked back to corresponding sanitized records at the subscriber in a privacy preserving manner.

Things become more challenging if this sharing needs to be carried out between multiple publishers and multiple subscribers. Publishers are often business competitors and unwilling to reveal to each other that they might be sharing clients between themselves. In such cases, two publishers should not know that they have a common group of clients. (Sharing such information under explicit directives from a client is allowed and is not considered here.) For privacy reasons, two subscribers should not be able to determine that they have clients in common; they should not be able to link or trace two sanitized records to the same client. When a publisher has more than one record for the same client, the same number of sanitized records should be available at the subscriber and be updated as needed. This occurs, for example, when a patient has repeated visits to the doctor for treatment.

Not much work has been done in this area of privacy preserving record linkage in dynamic setting. Some existing techniques that partially address the problem require encryption of linkage information using a shared key between data publishers to find if matched individuals' data exist across multiple sites. However, this technique works for small communities; it is expensive to deploy in large heterogeneous setups. In addition, shared keys among a large number of entities increase the chances of key leakage. An alternative technique that is used in the medical community is to utilize the services of a trusted third party, HB, called the Honest Broker. The third party maintains the linking information between the subscriber data and the publisher data in a de-identified manner. The problem with this approach is that the Honest Broker has all information, which makes it a lucrative target for attackers. If the Honest Broker is compromised it will cause a catastrophic damage to both data publishers as well as to individual clients.

In this work, we address this problem of privacy preserving record linkage by proposing a secure scheme based on partially homomorphic encryption and a third party. The third party is responsible just for automatically and blindly perform record matching. It is honest in the sense that it follows the protocol correctly but is not trusted to keep a secret, secret. It is curious about the sensitive information contained in individual records and can act accordingly.

However, our protocol ensures that it is prevented from getting such information without colluding with publishers. The third party knows that two publishers or subscribers have clients in common; however, it does not know the identities of these clients.

The main idea behind our protocol is as follows. Each data publisher creates a "key converter" in collaboration with other data publishers. Each key converter is then given to the third party (henceforth referred to simply as the broker). Each data publisher encrypts critical identification information of its data using it own secret key. Later, the broker uses the "key converters" to blindly transform all publisher-encrypted identification information to an alternate encrypted form under some other key that is not known to any party including the broker itself. The broker needs to collude with at least one of the publishers to find that key. Once the linkage information is encrypted under the same key, the broker can easily determine matching records. The broker can also keep track of the individuals found at different sites for retrospective update queries purposes. Since the data is encrypted at the source with different keys that the broker does not have access to, the risk of privacy breach in case of the broker getting compromised is limited.

The rest of the paper is organized as follows: In section 2, we walk through previous works related to the underlined problem. Section 3 gives some background information about primitives used to construct the scheme. In section 4, we detail our construction. In section 5, we provide a brief complexity and security analysis. Finally, section 6 concludes the paper.

2 Related Work

Privacy preserving data sharing has been a well studied problem, particularly in the context of sharing information from databases controlled by multiple parties. The participating parties in a privacy preserving database querying system are: the *data owner*, variously called the data source or publisher of the data, who provides access to its data to others, the *data querier* who generates query against the publisher's data and receives the results, and the *host* who (potentially) stores the publisher's data, and executes the query by performing relevant computations.

In our setting, we respectively denote by publisher, subscriber and broker the data owner, data querier and the host. The challenging scenario addressed in this work considers many competitor publishers that do not want to reveal any information about their data to each other but would like to anonymously and securely share some information with the subscriber. In addition, the subscriber is not only interested in querying their data separately, but jointly in order to find connected records across the databases. Furthermore, the subscriber wants to be able to retrieve updates about some previously queried entities.

If the querier is the owner of the data itself, and the host is an untrusted outsourced third party, searchable encryption schemes or Oblivious RAM (ORAM) are used to protect the data and maintain the owner privacy. Considerable

amount of research has been done towards this problem that resulted in very efficient and expressive schemes [1–6]. However, in searchable encryption schemes, the data to be joined must be encrypted under the same key that has been shared among the owners. So searchable encryption schemes cannot directly be applied for our scenario.

In some settings, there are at least two parties each hosting their own data and either one or both of them wants to query other's data against his data. Each of them is considered the adversary to the other and ideally should not learn any thing beyond the query result. For such cases, private set intersection [7, 8] and multi-party computation are potential solutions [9]. For instance, Agrawal et al. [10] propose a set of operations on private databases between two parties where nothing is learned by the parties rather than the results of the operations. De Cristofaro et al. [11] make also use of set intersection techniques, yet using a certification authority (CA) to authorize queries and leverage client and server privacy. However, these solutions although appealing from a security perspective are not very efficient for large settings.

Solutions proposed in [12, 13] introduce a third party (Honest Broker) who will work as a mediator between the querier and the owners. This solution is impractical and not secure. In fact, although these solutions could be easy to deploy, non-cryptographic hash usage makes unauthorized users able to compute the hash; if the domain of the information is small or well known then it is possible to find the original information or at least to verify if some value exists or not. If pseudo-random functions are used instead, then a shared secret key must be used, which is undesirable in our setup because of the competition between different publishers. In addition, owing to the significant amount of trust required on the Honest Broker it becomes an appealing target for the attackers and malicious insiders [14, 15].

Yin and Yau [16] propose a privacy preserving repository for data integration across data sharing services that allows owners to specify integration requirements and data sharing services to safeguard their privacy. The owners determine who and how their data can be used. The authors presented what they call context aware data sharing to help data services to share data with the repository. However, the matching process between records is done using the hash value of their identification information, which is not secure and does not preserve privacy. This scheme suffers from the same underlying problems of the previous construction, namely, a mandatory secret sharing between competing parties. Carbunar and Sion [17] introduce a mechanism for executing a general binary join operation on an outsourced relational database with low overhead using predefined binary finite match predicates for computed match set. However they assume that the keys of both columns used in the join are known to the data owner. If the data belongs to two different owners, this key must be shared beforehand – essentially the same problem as earlier.

Chow et al. [18] propose a model for performing privacy preserving operations in a distributed database environment. The model, called Two-Party Query computation, comprises a randomizer and computing engine that do not reveal

any information between themselves. The model in essence emulates a central party with the functions split into two entities. In order to guarantee that the randomizer and the computing engine do not collude, the authors proposes to use key agreement protocol among the participating entities. This protocol has limited applicability to our scenario since it doesn't not support the retrospective queries.

Finally, a solution presented by Tassa et al. [19] targets gathering data between horizontally or vertically divided dataset while preserving sensitive information. The computation is over sanitized data set using k-anonymity or l-diversity sanitizing techniques. Their main focus is on anonymization of distributed databases in such a way that each party locally anonymizes its data without revealing its original content to others. While the ideas and techniques of this work are suitable for distributed data mining problems, since our requirements are quite different Tassa et al.ś work cannot be easily used to serve our needs.

3 Background

In the following, we recapitulate some important concepts that we use to build our protocols.

3.1 ElGamal Cryptosystem

ElGamal public-key cryptosystem [20] is defined over finite field $\mathbb{F}_q$ of a prime order q. The public key pk equals (G, q, g, h), where G is a cyclic group of order q with g as a generator, and $h = g^x$. The private key sk equals $x \xleftarrow{R} \{1, \cdots, q-1\}$.

The encryption of a message m using the public key is (c_1, c_2) and computed such that $(c_1, c_2) = \mathsf{Enc}_{\mathsf{pk}}(m) = (g^r, m \cdot h^r)$, where $r \xleftarrow{R} \{0, \ldots, q-1\}$. To decrypt the ciphertext (c_1, c_2) and retrieve $m = \mathsf{Dec}_{\mathsf{sk}}(c_1, c_2)$, the private key is needed as follows: first compute $s = c_1^{\mathsf{sk}} = (g^r)^x$. The decrypted message m equals $m = c_2 \cdot s^{-1} = m \cdot h^r \cdot g^{-rx} = m \cdot (g^x)^r \cdot g^{-rx} = m$.

3.2 Elliptic Curve Cryptography and the Discrete Logarithm Problem (ECDLP)

An Elliptic Curve [21] E over a finite field $\mathbb{F}_q$ of prime order q, is a set of points with coordinates from that field defined by an equation of the form $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$ for all $a_i \in \mathbb{F}_q$, or of the simpler form $y^2 = x^3 + ax + b$, with $a, b \in \mathbb{F}_q$ for finite fields of order different from 2 or 3. The coefficients define the shape of the curve and are the parameters of the curve. The set of points together with a special point called the point at infinity $\mathcal{O}$, form a group under the addition of points operation. Multiplication of points is not defined; however, multiplying a point P by a scalar u is defined as the addition of the point P u number times, i.e. $uP = \underbrace{P + P + \cdots + P}_{u \text{ times}}$. The cyclic subgroup $E(\mathbb{F}_q)$ is defined

by its generator (base point) P with order n, which is the smallest positive integer such that $nP = \mathcal{O}$. This subgroup $E(\mathbb{F}_q) = \{\mathcal{O}, P, 2P, \ldots, (n-1)P\}$ is denoted by its domain parameters (q, a, b, P, n).

Given an Elliptic Curve $E(\mathbb{F}_q)$ over a finite field and two points $P, Q \in E$, it is hard to find an integer $x \in \mathbb{Z}_q$ such that $Q = xP$. This is known as the Elliptic Curve Discrete Logarithm Problem (ECDLP). ECDLP is believed to be harder than finding Discrete Logarithms for finite fields, which is why many public key cryptosystems uses Elliptic Curves (EC) as the underlying group. For our purposes, the ElGamal cryptosystem and its variations are defined using EC as follows.

First, the communicating parties agree on the EC parameters and the corresponding field, i.e., $E(\mathbb{F}_q)$, the generator point G, and its order n. From these parameters each party generates its private key as $\mathsf{sk}_i = x, x \xleftarrow{R} \{1, \cdots, n-1\}$, and its public key as the point $\mathsf{pk}_i = xG$. The messages to be encrypted must be encoded as points on the curve in order to apply the group addition, or the messages must be integer scalars in the range $\{1, \cdots, n-1\}$ to use multiplications of points by scalars. The encryption of encoded message M is then performed by the sender A using the recipient B's public key sk_B as $C_1 = r \cdot G$, and $C_2 = r \cdot \mathsf{pk}_B + M$, where $r \xleftarrow{R} \{1, \cdots, n-1\}$. The decryption at the receiver side B is done using its private key sk_B as $M = C_2 - \mathsf{sk}_B \cdot C_1 = r \cdot \mathsf{pk}_B + M - sk_B \cdot r \cdot G = r \cdot k \cdot G - k \cdot r \cdot G + M = M$.

3.3 Homomorphic Encryption

Homomorphic encryption allows arithmetic operations to be carried out on ciphertext in such a way that the decrypted result matches the result of the operations when performed on the plaintext. *Partially* Homomorphic Encryption system (PHE) allows either addition or multiplication but not both. In this work, we focus only on the multiplicative property.

The homomorphic property of ElGamal cryptosystem over a finite field ensures that the product of two encrypted messages $\mathsf{Enc}_{\mathsf{pk}}(m_1)$ and $\mathsf{Enc}_{\mathsf{pk}}(m_2)$ will decrypt to the product of their corresponding plaintext messages $m_1 \cdot m_2$,

$$\begin{aligned}
\mathsf{Enc}_{\mathsf{pk}}(m_1) \cdot \mathsf{Enc}_{\mathsf{pk}}(m_2) &= (g^{r_1}, m_1 \cdot h^{r_1}) \cdot (g^{r_2}, m_2 \cdot h^{r_2}) \\
&= (g^{r_1+r_2}, (m_1 \cdot m_2) h^{r_1+r_2}) \\
&= \mathsf{Enc}_{\mathsf{pk}}(m_1 \cdot m_2).
\end{aligned}$$

If $s = c_1^{\mathsf{sk}} = g^{(r_1+r_2)x}$ then $s^{-1} = g^{-(r_1+r_2)x}$ and the decryption will result in

$$\begin{aligned}
\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(m_1) \cdot \mathsf{Enc}_{\mathsf{pk}}(m_2)) &= \mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(m_1 \cdot m_2)) \\
&= \mathsf{Dec}_{\mathsf{sk}}(g^{r_1+r_2}, (m_1 \cdot m_2) h^{r_1+r_2}) \\
&= (m_1 \cdot m_2) \cdot g^{(r_1+r_2)x} \cdot s^{-1} \\
&= (m_1 \cdot m_2) \cdot g^{(r_1+r_2)x} \cdot g^{-(r_1+r_2)x} \\
&= m_1 \cdot m_2
\end{aligned}$$

4 Scheme Construction

Our scheme works in three phases: the *setup* phase, the *encryption of query results* phase, and the *secure record matching* phase. The setup phase is executed only once by the data publishers to create the so-called publishers' "key converters". It utilizes properties of homomorphic encryption to create these "key converters". The encryption of query results phase occurs at the publisher side whenever it executes a query, to encrypt the identifying part of the query results. This encryption is performed using the publisher's secret non-shared key before sending the results to the broker. The secure record matching phase occurs at the broker side to determine the linkages between the records coming from the publishers after executing the queries.

We begin by discussing the adversarial model and privacy requirements, then we explain each phase in details.

4.1 Adversary model and privacy requirements

Data publishers are considered competitors who do not want to share data with each other. Each publisher tries to determine information about the clients of competitor publisher, or at a minimum, tries to determine if any one of its clients is also a client of its competitor. However, publishers are also honest in the execution of the protocol steps and willing to share information with subscribers *privately*, that is, without revealing real identities, and *securely*, that is, without any leakage of information to other data publishers, and without revealing the publisher identity to the subscribers.

A data subscriber, on the other hand, needs to determine if any information that came from different publishers belong to the same individual so they could be grouped together as such and treated accordingly. For example, if a researcher is looking for the side effects of a new drug used for skin treatment on patients who has kidneys problems, then he has to match patients from the (Dermatology) and (kidney diseases) departments to find patients under these conditions. We need to allow such grouping at the subscriber side.

Further more, the subscriber is allowed to issue *retrospective* queries regarding some individual client, for example, update queries regarding the progress of treatment of certain patients. Subscribers (researchers) are considered curious in the sense they will try to determine the real identities of the individuals. Some information about individual identities might be leaked from their non-identification information (i.e. eye color, age, weight, etc.) using statistical inference techniques. This is a separate problem that needs to be addressed with anonymization (i.e. k-anonymity) or other sanitization methods, and is not considered in this work.

The broker is honest in the sense that it will not collude with any of the parties, but is curious and not trusted to keep a secret, secret. The broker will work as a mediator between the data publishers and the subscribers by honestly performing the following tasks:

– Hide the source of information (publishers and clients' identities) from the subscribers.
– Blindly determine record linkages among the encrypted publishers' records and assign alternate random identifiers to the linked records before sharing them with the subscribers. The broker will just know the linkages without knowing the real identifiers.

4.2 Setup phase

In order to create its key converter, each publisher is required to initially interact with other publishers participating in the sharing system, keeping in mind that other publishers are competitors. Every publisher needs to go through the setup protocol once at the beginning of the system instantiation when it joins the sharing system. An existing publisher also needs to embark upon the setup phase if a refreshing of keys is required when new publishers join the system. These key converters will be delegated to the third party (broker) and will be used to convert records encrypted under different keys of different publishers, to records encrypted under a common key. This common key is such that it cannot not be re-constructed by any of the parties, namely, individual publishers, broker and subscribers. This means that the encrypted data is safe if there is no collusion between any of the publishers and the broker at the instantiation time. In this scheme, we propose a secure protocol to create the key converters among the publishers using ElGamal homomorphic cryptosystem that supports product operation over the encrypted keys. At the end of this phase, every publisher is associated with a special key converter that allows the broker to perform the matching process.

Each publisher from the set of N publishers $D = \{d_i\}_{i\in[N]}$ has its secret key $\mathsf{sk_i}$, and the broker which has a public key pk and private key sk pair. The setup phase is illustrated in Fig.1 and works as follows.

– **Step 1**: The broker broadcasts its public key $\mathsf{pk} = (G, q, g, g^{\mathsf{sk}})$ for ElGamal homomorphic encryption to all publishers.
– **Step 2**: Each publisher d_i generates an initial random secret $r_i \xleftarrow{R} \{1, \cdots, q-1\}$, where q is the order of G, encrypts it using the master key pk. Let $t_{i\to j}$ denote the temporary encrypted key converter of publisher i when being processed by publisher j, and $t_{i\to final}$ the final converter when it gets back to the publisher i after being processed by all parties. Publisher d_i generates the initial temporary encrypted key converter $t_{i\to i} = \mathsf{Enc_{pk}}(r_i^{-1})$, then forwards it to the next publisher $d_{(i \mod (N))+1}$.
– **Step 3**: Each publisher d_j receives a value $t_{i\to j}$ from its upstream neighbor d_i $(i \neq j)$, securely multiplies it using ElGamal homomorphic cryptosystem with its secret key $\mathsf{sk_j}$ encrypted under the broker's public key pk as follows:

$$\begin{aligned} t_{i\to i+1} &= t_{i\to i} \cdot \mathsf{Enc_{pk}}(\mathsf{sk_{i+1}}) \\ &= \mathsf{Enc_{pk}}(r_i^{-1} \cdot \mathsf{sk_{i+1}}) \end{aligned}$$

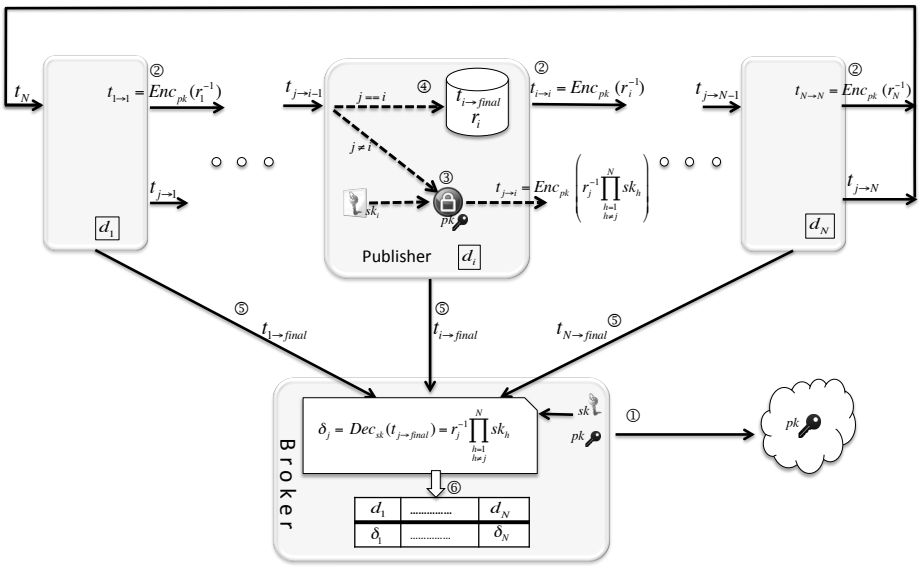

Fig. 1. High level setup phase illustration

This operation is repeated through $N-1$ publishers. The temporary value of publisher d_i generated by the j^{th} publisher equals:
$t_{i\to j} = \mathsf{Enc_{pk}}(r_i^{-1} \cdot \prod_{c=i+1}^{j} \mathsf{sk_c})$

– **Step 4**: After $N-1$ transitions, the initials publisher receives the final key converter $t_{i\to final}$ as

$$t_{i\to final} = \mathsf{Enc_{pk}}\left(r_i^{-1} \prod_{\substack{j=1 \\ j\neq i}}^{N} \mathsf{sk_j}\right)$$

– **Step 5**: After each publisher is able to generate its key converter value after being processed by all other publishers, the publisher d_i sends its $t_{i\to final}$ value to the broker, and saves a copy of it for future updates in case new publishers joined the system.
– **Step 6**: For each $t_{i\to final}$, the broker extracts the key conversion factor δ_i such that: $\delta_i = \mathsf{Dec_{sk}}(t_{i\to final}) = r_i^{-1} \prod_{\substack{j=1 \\ j\neq i}}^{N} sk_j$

Key converter refresh: If a new publisher d_{N+1} joins the system, then it follows the previous steps to create its own key converter t_{N+1}, while other publishers just need to update their own key converters. To accommodate the key of the new publisher, old publishers send their previously created $t_{i\to final}$ values to the new one, which in turn securely adds its secret key sk_{N+1} to this value $t_{i\to final}$ to get $t'_{i\to final}$, and sends it back to the source. Each publisher d_i

then refreshes its $t'_{i\to final}$ with a new random value r'_i and sends the updated $t'_{i\to final}$ to the broker. This new random value r'_i is used to prevent the broker from extracting information about the newly added secret key of the new party d_{N+1} by comparing the new with the old $t_{i\to final}$ values. Each publisher updates its secret key with this new random value too.

Note: Careful readers might think that it would be simpler if publishers broadcast their keys and then locally compute the key converters. It is true that in term of communication overhead, this method also involves $O(n^2)$ interactions, however, in term of information shared, it leaks more. In this solution, each publisher submits its key encrypted with the public key of the broker. If the broker can somehow eavesdrops the communication between the publishers, it can decrypt and obtains the key of all publishers.

4.3 Encryption of query results phase

This phase is triggered by a query sent by a subscriber requesting information from publishers. This represents a *data pull* model; however, our scheme can be also used in a *data push* mode where publishers send data directly to the broker which redirects it to the corresponding subscribers. After executing the received query, each publisher encrypts the identifying parts of the query results using any cryptosystem that relies on DDH (Decisional Diffie-Hellman) or DL (Discrete Logarithm) hardness assumptions, such as ElGamal cryptosystem. For performance reasons, our construction uses Elliptic Curves (EC) instead of Finite Groups of large primes as the underlying group for the used cryptosystem.

Each publisher has the publicly known ElGamal EC parameters, i.e., the curve parameters $E(\mathbb{F}_q)$ and the point on the curve P of prime order n. The public/private key pair will be $(r_i \cdot \mathsf{sk}_i \cdot P, r_i \cdot \mathsf{sk}_i)$ and both of th keys are kept secret. The message to be encrypted, id, in our multiplicative scheme needs to be a scalar. We denote by $\mathsf{E}(.)$ the encryption of ElGamal based on EC.

The publisher first hashes the identifying part of every record in the result set using a universal hash function H. The result set is the data outputted by executing the subscriber query. The publisher uses its secret key multiplied by the corresponding random value, $(r_i \cdot \mathsf{sk_i})$, to encrypt the resulting hash. That is, the encryption of any identifier id will be:

$$\mathsf{E}_{(r_i \cdot \mathsf{sk_i})}(H(id)) = H(id) \cdot r_i \cdot \mathsf{sk_i} \cdot P$$

Finally, the publisher substitutes the real identifying part, id, by $\mathsf{E}_{(r_i \cdot \mathsf{sk_i})}(H(id))$ for all records in the result set. Finally, each record is composed of the encrypted identification part, plus, the other client's information. The data in plaintext in each record will be sanitized if necessary, according to the publisher's policy, before being sent to the broker. Sanitizing techniques details are out of scope of this work.

Publishers can avoid having the broker store the key converter $(\delta_i)_{i\in[N]}$. For this purpose, each publisher encrypts the identifiers of the query results with a new random value r_i, updates the key converter $t_{i\to final}$, then sends these

results to the broker accompanied with the new key converter. This solution adds negligible communication overhead, but ensures a zero-key stored on the broker side.

4.4 Secure record matching phase

The broker receives the encrypted identifiers with different keys from different publishers. The broker's job is to merge similar clients' records from different publishers such that they will map to the same newly generated identifier. The broker will use the key converters δ_i to change the encryption key in such a way that similar data will be deterministically encrypted with the same key without requiring any decryption to be performed along the way.

The broker uses the δ_i values to convert any identifier id encrypted by publisher d_i under its secret key $(r_i \cdot \mathsf{sk}_i)$, to a value encrypted under a different secret key Δ, i.e., $\mathsf{E}_{\Delta}(H(id))$. The key $\Delta = \prod_{i=1}^{N} \mathsf{sk}_i$ is resulting from the product of all the secret keys of all publishers. In order to perform the secure record matching, the broker re-encrypts the encrypted identifying parts of the records coming from the publisher d_i using the corresponding key converter δ_i as:

$$\mathsf{E}_{\delta_i}\left(\mathsf{E}_{(r_i \cdot \mathsf{sk}_i)}(H(id))\right) = \mathsf{E}_{\Delta}(H(id))$$

That this process does indeed perform correct matching is shown by the fact that:

$$\begin{aligned}
\mathsf{E}_{\delta_i}\left(\mathsf{E}_{(r_i \cdot \mathsf{sk}_i)}(H(id))\right) &= \mathsf{E}_{\delta_i}\left(H(id) \cdot r_i \cdot \mathsf{sk}_i \cdot P\right) \\
&= H(id) \cdot r_i \cdot \mathsf{sk}_i \cdot P \cdot r_i^{-1} \prod_{j=1, j\neq i}^{N} \mathsf{sk}_j \\
&= H(id) \cdot \prod_{j=1}^{N} \mathsf{sk}_j \cdot P \\
&= H(id) \cdot \Delta \cdot P = \mathsf{E}_{\Delta}(H(id))
\end{aligned}$$

In order to maintain the linkages between publishers' data records and the randomly generated identifiers for subscribers, the broker keeps track of the processed identifiers for both flows, i.e., from publishers to subscribers and vice versa. The aim of this mapping is two folds: first we do not want to give the ability to the subscribers to know whether they share the same client and second give the ability to the broker to map back these random values to the same client. For this purpose, the broker builds two secure inverted indexes, one to map the encrypted identifiers after conversion (i.e. $\mathsf{E}_{\Delta}(H(id))$) to their corresponding subscriber identifiers such that for each we generate a random value rid concatenated to the subscriber identifier sid. The second maps $rid \| sid$ to the set of corresponding encrypted clients' identifiers concatenated with their publisher id such that $\mathsf{E}_{\mathsf{r_i} \cdot \mathsf{sk_i}}(H(id)) || d_i$, for $i \in [N]$, see Table. 1. These secure inverted indexes can be constructed following searchable encryption data structure instantiation, see [1, 2].

Table 1. Conceptual representation of secure inverted indexes

$\mathsf{E}_{\Delta}(H(id))$	$rid \| sid$
0xAB4542..24	0x354AE2..16 ‖ 1, 0xF14598..24 ‖ 5
0xC2C6A5..59	0x413F56..AE ‖ 2
.....	..

$rid \| sid$	$\mathsf{E}_{r_i \cdot \mathsf{sk}_i}(H(id)) \| \| d_i$
0x354AE2..16 ‖ 1	0x6547A..6A ‖ 2, 0x45CA4..B2 ‖ 5
0x413F56..AE ‖ 2	0x48F53..12 ‖ 11
..	.....

4.5 Matching phase walk-through

We now summarize all the interactions between the three parties, namely, publishers, subscribers and the broker to describe how privacy preserving record matching system works to serve the subscriber's needs. These interactions schematically shown in Fig. 2, with each of following steps corresponding to the numbers shown in the figure.

1. The subscriber sends a query Q to the broker.
2. The broker sanitizes the query if necessary, and checks if this query Q is a query that seeks information about new clients or a retrospective query (requesting more information about an individual whose data has been seen before). If it is a new query, it forwards the query to publishers and wait for the answers.
3. If the query Q is a retrospective query, the broker first looks up the $rid \| sid$ in the inverted index for the corresponding encrypted identifiers and their associated publisher identities, i.e. $\mathsf{E}_{\mathsf{r_i} \cdot \mathsf{sk_i}}(H(id)) \| \| d_i$. The broker then replaces $rid \| sid$ with the corresponding $\mathsf{E}_{\mathsf{r_i} \cdot \mathsf{sk_i}}(H(id))$ and finally forwards the query Q only to their associated publishers d_i.
4. For each query Q it receives, the publisher d_i checks if it is a new query or a retrospective query. If it is a retrospective query, the publisher can directly look up the matching records while for a new query a search has to be done to find the corresponding clients. In either case, the publisher applies its local sanitizing policies to the results, encrypts the identification part using its secret key $(r_i \cdot \mathsf{sk}_i)$, and finally sends the results to the broker.
5. Upon receiving the results of the forwarded query from the publishers, the broker further sanitizes the results according to the general sharing policies and regulations. Then it performs the secure record matching process, and updates its inverted indexes as follows:
   - Using the first inverted index, and for each distinct $\mathsf{E}_{\mathsf{r_i} \cdot \mathsf{sk_i}}(H(id))$ in the query result, apply the key converters to get $\mathsf{E}_{\Delta}(H(id))$. If a match is found then this identifier has been sent before (it might be from a different publisher though). So the broker retrieves the corresponding $\{rid \| sid\}$, and updates the second inverted index with the encrypted

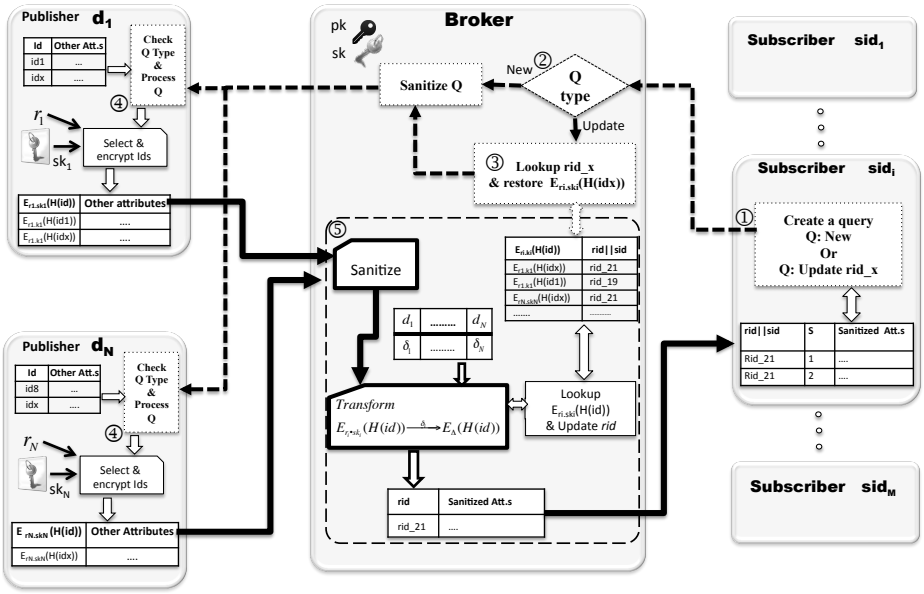

Fig. 2. Matching phase walk-through illustration

id $\mathsf{E}_{\mathsf{r_i}\cdot\mathsf{sk_i}}(H(id))||d_i$ in case it has been previously sent from a different publisher.

- If the converted encrypted $\mathsf{E}_\Delta(H(id))$ is not found, then it means that this identifier has not been sent before by any publisher. The broker adds this converted value to the first inverted index, with a new subscriber and random identifier $rid\|sid$. The second inverted index is updated accordingly.
- The encrypted identifier $\mathsf{E}_{\mathsf{r_i}\cdot\mathsf{sk_i}}(H(id))$ in each record in the results is then replaced with its corresponding $rid\|sid$ before being sent to the corresponding subscribers.

As we have mentioned earlier, these inverted indexes are encrypted under the broker's secret keys, and all the searching and update operations are performed on encrypted data using any adaptive secure searchable encryption scheme data structures.

5 Complexity and Security Analysis

Complexity: Our scheme is practicable and can be efficiently implemented in real scenarios. The construction is composed of three main steps, the setup, the encryption of query results and the secure record matching phase. The setup phase depends on the number of publishers N. The entire setup phase results in a total communication overhead which is $O(N^2)$. To generate *one* key converter, the publishers needs to transfer one message each, while the computation is

constant per each transfer. The total communication and computation overhead per publisher is in $O(N)$. The setup phase is performed just once and is done in an off-line manner. The encryption of query results overhead depends on the matching set and the data structure that the publisher is using for its own data storage. The search complexity depends on the query sent by the subscriber through the broker. Also the complexity of this phase greatly depends on the size of the database of the publishers and the sanitizing protocols used as well. For this reason, we consider only the communication overhead for this phase. For the last phase performed by the broker, given all identifiers that match the query, the goal consists of retrieving all subscribers identifiers as well as the randomized identifiers. Using SSE, the search is optimal and similar to plaintext data. Given an identifier, the search complexity will be in the number of matching results.

To sum up, the construction does not induce any overhead during the matching phase more than the one expected on plaintext data. The linear construction of the key converters is done once during the instantiation of the protocol.

Security: The key converter is based on ElGamal homomorphic encryption which is semantically secure. However, we want to make sure that the output of the key converter received by the broker will not leak any information about the secret keys of any publisher. The broker receives N key converters for N different publishers. Every key converter is composed of $N-1$ secret key and a random value. The broker then has, at the end of the setup phase, a linear system of N equations with $2N$ unknowns. This is an under-determined system which is information theoretically secure. This property ensures that the broker cannot, based on the key converters, recover the secret keys of the publishers. Note that this is true as long as the publishers do not collude with the broker. We can enhance this key converter generation with much secure schemes such as multi-party computation where we can take into consideration malicious parties.

During the encryption of query results phase, every publisher is encrypting the records with a different secret key. Thus, even if a publisher eavesdrops over the communication between publishers and broker, it cannot infer anything about the clients identity.

For the storage of the encrypted identifiers at the broker side, a conversion of the encryption is necessary. The broker therefore knows that this encrypted identifier (whatever be the identifier) is linked to the same client in different publishers. This feature is very important for the scheme correctness since it enables the broker to aggregate the exact subscriber query results. From a security perspective, we want to hide not only the association between encrypted identifiers and subscribers but also the mapping between the publisher identifiers and the encrypted identifiers. For this purpose, we use a symmetric searchable encryption inverted index that enables to store securely these mappings. Consequently, even if the data at the broker is somehow leaked, the entire inverted indexes are encrypted and the mapping will not be disclosed.

6 Conclusions and Future Work

We have presented a novel scheme to allow multiple data publishers, who encrypt their data with their private non-shared key, to share their clients' records with subscribers while protecting the identities of their clients. The scheme is based on partially homomorphic encryption to create "key converters" that allow a third party with limited trust (broker), to perform privacy preserving record matching. The role of the broker is to determine the linkages without identifying the real identities of the publishers' clients but also to hide from the subscriber which publisher generated the information. Our construction is achieved by allowing each publisher to use his own secret key to guarantee the security of its data, and the broker, with the help of the key converters, to convert encrypted data of each publisher under different keys to data encrypted under the same key. Both the key converter and data conversion are implemented using ElGamal cryptosystem. Future work aims to provide an experimental evaluation with real-world clinical data and a practical implementation of this system for the clinical data-sharing network called CHORDS (Colorado Health Outcome Research Data Services).

Acknowledgment

This work was partially supported by the U.S. National Science Foundation under Grant No. 0905232, and by Colorado State University under an internal research grant.

References

1. Chase, M., Kamara, S.: Structured encryption and controlled disclosure. In: Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore (2010) 577–594
2. Curtmola, R., Garay, J.A., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: Proceedings of the 13th ACM Conference on Computer and Communications Security, Alexandria, VA, USA (2006) 79–88
3. Moataz, T., Shikfa, A.: Boolean symmetric searchable encryption. In: Proceedings of the 8th ACM Symposium on Information, Computer and Communications Security, Hangzhou, China (2013) 265–276
4. Cash, D., Jarecki, S., Jutla, C.S., Krawczyk, H., Rosu, M., Steiner, M.: Highly-scalable searchable symmetric encryption with support for Boolean queries. In: Proceedings of Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA (2013) 353–373
5. Stefanov, E., van Dijk, M., Shi, E., Fletcher, C.W., Ren, L., Yu, X., Devadas, S.: Path ORAM: an extremely simple oblivious RAM protocol. In: Proceedings of ACM Conference on Computer and Communications Security, Berlin, Germany (2013) 299–310

6. Strizhov, M., Ray, I.: Multi-keyword similarity search over encrypted cloud data. In: Proceedings of 29th IFIP TC 11 International Conference, Marrakech, Morocco (2014) 52–65
7. Dachman-Soled, D., Malkin, T., Raykova, M., Yung, M.: Efficient robust private set intersection. International Journal of Applied Cryptography **2** (2012) 289–303
8. Kamara, S., Mohassel, P., Raykova, M., Sadeghian, S.S.: Scaling private set intersection to billion-element sets. In: Proceedings of the 18th International Conference of Financial Cryptography and Data Security, Christ Church, Barbados (2014) 195–215
9. Goldreich, O.: Secure multi-party computation. Manuscript. Preliminary version (1998) http://citeseerx.ist.psu.edu, accessed 30-April-2015.
10. Agrawal, R., Evfimievski, A., Srikant, R.: Information sharing across private databases. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, CA, USA (2003) 86–97
11. De Cristofaro, E., Lu, Y., Tsudik, G.: Efficient techniques for privacy-preserving sharing of sensitive information. In: Proceedings of the 4th International Conference on Trust and Trustworthy Computing, Pittsburgh, PA, USA (2011) 239–253
12. Boyd, A.D., Saxman, P.R., Hunscher, D.A., Smith, K.A., Morris, T.D., Kaston, M., Bayoff, F., Rogers, B., Hayes, P., Rajeev, N., Kline-Rogers, E., Eagle, K., Clauw, D., Greden, J.F., Green, L.A., Athey, B.D.: The University of Michigan honest broker: A web-based service for clinical and translational research and practice. Journal of the American Medical Informatics Association : JAMIA **16** (2009) 784–791
13. Dhir, R., Patel, A.A., Winters, S., Bisceglia, M., Swanson, D., Aamodt, R., Becich, M.J.: A multidisciplinary approach to honest broker services for tissue banks and clinical data. Cancer **113** (2008) 1705–1715
14. Jefferies, N., Mitchell, C.J., Walker, M.: A proposed architecture for trusted third party services. In: Proceedings of the International Conference on Cryptography: Policy and Algorithms, Brisbane, Queensland, Australia (1995) 98–104
15. Ajmani, S., Morris, R., Liskov, B.: A trusted third-party computation service (2001) http://www.pmg.lcs.mit.edu/∼ajmani/papers/tep.ps, accessed 30-April-2015.
16. Yau, S., Yin, Y.: A privacy preserving repository for data integration across data sharing services. IEEE Transactions on Services Computing **1** (2008) 130–140
17. Carbunar, B., Sion, R.: Toward private joins on outsourced data. Knowledge and Data Engineering, IEEE Transactions on **24** (2012) 1699–1710
18. Chow, S.S., Lee, J.H., Subramanian, L.: Two-party computation model for privacy-preserving queries over distributed databases. In: Proceedings of the 2009 Network and Distributed System Security Symposium, San Diego, CA, USA (2009)
19. Tassa, T., Gudes, E.: Secure distributed computation of anonymized views of shared databases. ACM Transactions on Database Systems (TODS) **37** (2012) 11
20. Gamal, T.E.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Proceedings of Advances in Cryptology - CRYPTO 1984 - 14th Annual Cryptology Conference, Santa Barbara, California, USA (1984) 10–18
21. Koblitz, N.: Elliptic curve cryptosystems. Mathematics of computation **48** (1987) 203–209