

Privacy Preserving Integration of Health Care Data

Nabil Adam, PhD¹, Tom White, MD², Basit Shafiq, PhD¹, Jaideep Vaidya, PhD¹, Xiaoyun He¹
¹Rutgers University, Newark, NJ; ²NY Office of Mental Health & Columbia University, NYC

Abstract

For health care related research studies the medical records of patients may need to be retrieved from multiple sites with different regulations on the disclosure of health information. Given the sensitive nature of health care information, privacy is a major concern when patients' health care data is used for research purposes. In this paper, we propose an approach for integration and querying of health care data from multiple sources in a secure and privacy preserving manner.

Introduction

Health care research studies often involve analysis of huge amount of data collected from various sources including health care providers, pharmacies, insurance companies, government agencies, and research institutions. Given the sensitive nature of health information and the social and legal implications for its disclosure, privacy is a major concern for information sharing in the healthcare domain [1,2,3].

Protecting the privacy of individually identifiable health information is more important when such information is used for clinical or health services related research. The Health Insurance Portability and Accountability Act (HIPAA) privacy rule strictly prohibits sharing of individually identifiable health information with clinical researchers who are not covered entities. The covered entities, as defined in this privacy rule, include health plans, healthcare clearing houses, and healthcare providers that transmit health information electronically in connection with certain defined HIPAA transactions, such as claims or eligibility inquiries [2].

For preventing disclosure of individually identifiable information, usually de-identified or anonymized health data are shared with researchers. This data may be retrieved from multiple sites with different regulations on the disclosure of health information. In absence of the identity information, correlation and integration of health data on a per patient basis in a privacy preserving manner is challenging issue. As an example, consider the following queries related to a research study of anti-depressant drugs:

Query 1: What percentage of HIV infected patients taking any prescribed anti-depressant medication are diagnosed with an acute psychiatric disorder?

Query 2: For each HIV infected patient diagnosed with acute psychiatric disorder find all the prescribed drugs the patient took after being diagnosed with HIV.

The above queries require integrating data from multiple sources including the state health department managing HIV test records, pharmacy databases storing patient records related to prescription drugs, and mental health clinics treating patients with psychiatric disorder. For preserving the privacy of individual records, we need to ensure that the query result do not reveal any individually identifiable information to the querying party. Additionally, during the process of integrating data from multiple sources, none of the sources should be able to learn/infer any information about any of the patients beyond what these sources already know. For instance, the pharmacist should not be able to learn which patients have been tested positive for HIV or which patients are receiving treatment for severe psychiatric disorder.

One of the large challenges in merging data is the lack of a common identifier across data systems. There are numerous commercial applications for creating enterprise master person indexes from distributed databases. Some of these are tolerant of missing, mistyped, or conflicting data. The approach commonly used by Regional Health Information Organizations (RHIOs) is that of RxHub – to only cross-link patients who have exact matches of five elements: first name, last name, birthday, gender, and zip code. RxHub is a third party data aggregator which has contracts with most major pharmacy benefits managers. Hospitals may use RxHub to request all historical pharmacy data from patients by sending those five demographic parameters to RxHub. RxHub queries its data sources to determine whether they have data for the requested patient, and if so, retrieve and organize the data from the various sources, keeping appropriate audit trails.

For now, this paper focuses on the challenges of research use of patient data, and assumes that all data sources have the same, shared, complete identifier, like social security number. Thus, assuming that the patients in each source's database are identified by their *social security number* (SSN), the data from multiple sources can be joined on this attribute to answer the above queries. However, this simple joining of data would reveal personally identifiable

health information to the querying party as well as to the data sources. Of course, one may trust a third party (like RxHub) to have access to all of the data and perform the join [3]. However, this is not sufficient. It would be much better to reduce the trust factor as much as possible.

We propose an approach that allows querying and integration of data from multiple sources. The proposed approach ensures secure and privacy preserving data integration in a semi-honest environment, where each party (data source and query party) is assumed to follow the protocol correctly. The proposed approach employs a cryptography based solution, whereby all the sensitive values in the query result are encrypted by all the data sources using their own keys. Since the encryption key of a source is not known to any other source or the querying party, therefore it is computationally infeasible for any party to extract the individually identifiable or sensitive information from the query result. Moreover, the proposed approach does not allow the querying party to retrieve any sensitive or non-sensitive information from sources which is beyond the scope of the query.

Related Work

This paper builds on our earlier work on privacy-preserving association rule mining using commutative encryption [6]. Recently, many commutative encryption-based approaches have been proposed for secure information sharing. Agarwal et al [4] propose a similar approach for information sharing across databases. This approach can be used to answer queries like Query 1, however, the approach is restricted to only two parties. Also, another problem is that no third party is used (i.e., the querying party could actually contribute data) – but this can lead to severe problems with security if the querying party is malicious. The querying party will also learn the values of all the attributes in the query result. If the query result is computed by joining data from different sources on the SSN field, then the querying party will learn the identity of patients from the query result.

A similar commutative encryption based approach is proposed by Malin and Sweeney [7] for sharing de-identified healthcare data that is *k-anonymous* and cannot be linked to any publicly available data to identify individual patients. However, they do not address the issue of data integration in [7].

Another work closely related to ours is the protocol by O'Keefe et al [5]. While the key concepts are similar, there are two problems. First, it only works for two data sources. More importantly, a malicious

querying party can trick the data source and get extra information which is not included in the query result.

Problem Definition

We consider a distributed environment with heterogeneous distribution of data. This means that different sources collect different features of information for the same set of data. As opposed to this, with homogeneous data distribution, different sources collect the same pieces of information about different entities. The second case is easier to deal with since there is no real integration to be done. Therefore we only look at the first case. Each source S_i has a relation R_i with attributes $(a_{i1}, a_{i2}, \dots, a_{ik})$, where a_{i1} uniquely identifies tuples in the relation R_i , meaning that a_{i1} is the primary key of the relation R_i . Without loss of generality we assume that no other attribute or set of attributes excluding a_{i1} can serve as a candidate key for the relation R_i . We assume that the primary key of each relation is the same and the join of two or more relations is always computed on the primary key value, i.e., only those tuples in the relation R_i and R_j can be joined for which $a_{i1} = a_{j1}$. We say that the relation with attributes $(f(a_{i1}), a_{i2}, \dots, a_{ik}, a_{j2}, \dots, a_{jm})$ is a privacy preserving join of relations R_i and R_j if $a_{i1} = a_{j1}$ and it is computationally infeasible for any party including data sources to determine a_{i1} from $f(a_{i1})$.

Thus, in this paper, we are addressing the problem of computing the privacy preserving join of n relations; whereby a relation R_i ($1 \leq i \leq n$) is owned by source S_i .

Proposed Approach

The basic idea of the proposed approach is to use commutative encryption to encrypt all data items in each party's data set. Commutative encryption is an important tool used in many cryptographic protocols. An encryption algorithm is commutative if the order of encryption does not matter. Thus, for any two encryption keys E_1 and E_2 , and any message m , $E_1(E_2(m)) = E_2(E_1(m))$. The same property applies to decryption as well – thus to decrypt a message encrypted by two keys, it is sufficient to decrypt it one key at a time. The basic idea is for each source to encrypt its data set with its keys and pass the encrypted data set to the next source. This source again encrypts the received data using its encryption keys and passes the encrypted data to the next source until all sources have encrypted the data. As discussed in the above section, the attributes of a data set can be divided into two: key attribute and non-key attributes. The key attribute of each data set is the same and the join of two or more data sets from different sources is always computed on the values of their key attribute. Since we are using commutative

encryption, the encrypted values of the key attribute across different data sets will be equal if and only if their original values are equal. Thus, all the data sets from different sources can be joined on their key attributes which are encrypted by all sources. The encryption prevents any source or querying party from knowing the actual value of the key attribute.

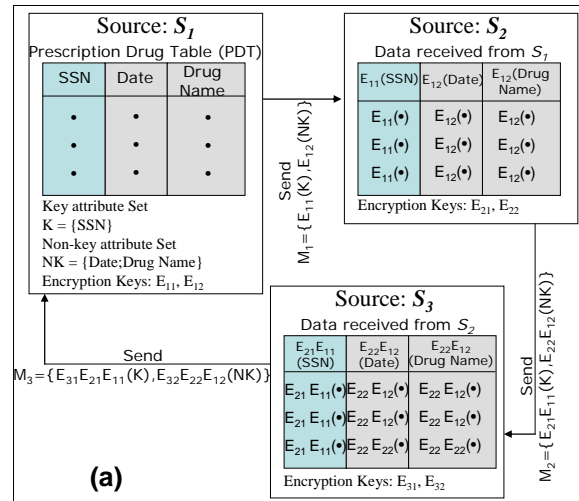
The algorithm for computing the privacy preserving join of data sets from multiple sources consists of three stages: *encryption*, *joining* and *decryption*. In the following we discuss each of these stages.

Encryption. In this stage, the joining attribute of each data set is encrypted by all sources. Each source S_i generates two pairs of encryption and decryption keys: (E_{i1}, D_{i1}) and (E_{i2}, D_{i2}) . For each data set R_i ($1 \leq i \leq n$), the source S_i encrypts the values of the key attribute and non-key attributes of all tuples in the data set R_i with its keys E_{i1} and E_{i2} respectively. After this encryption, the source S_i passes the resulting data set $(E_{i1}(a_{i1}), E_{i2}(a_{i2}), \dots, E_{i2}(a_{ik}))$ to its right neighbor S_j , where $j = (i+1) \bmod n$. The neighbor S_j in turn encrypts the received data set with its encryption keys E_{j1} and E_{j2} . In addition S_j permutes the tuples in the encrypted data set to prevent the owner of the data set to correlate the encrypted values of the key attribute with their actual values based on the ordering of tuples in the data set. S_j then forwards the resulting data set to its next neighbor for encryption. This process continues until the data set is encrypted by all the sources.

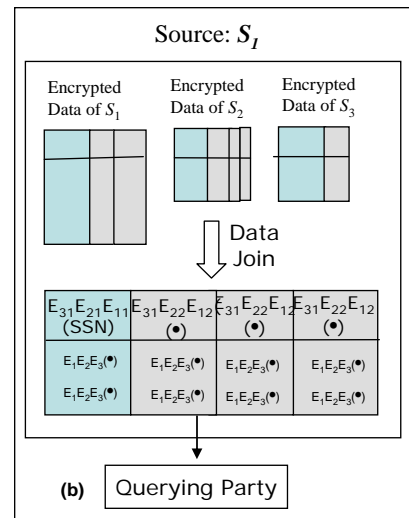
Figure 1(a) illustrates the data flow and message exchange among three sources S_1 , S_2 , and S_3 for encryption of *prescription drug data* managed by S_1 .

Joining. Once all the data sets are encrypted by all sources, they can be joined on the encrypted value of the key attribute as discussed above. This join can be computed by any source which has all the encrypted data sets. Once the source for computation of the final join has been selected, each source will forward its local data set which is encrypted by all other sources to the selected source. The selected source then computes the join and sends the resulting data set to the querying party. We will refer to the resulting data set after joining tuples from different data sets on the encrypted value of their key attributes as the *joined data set*. Note that it is computationally infeasible for any source or the querying party to extract the original value of any of the attributes of the joined data set.

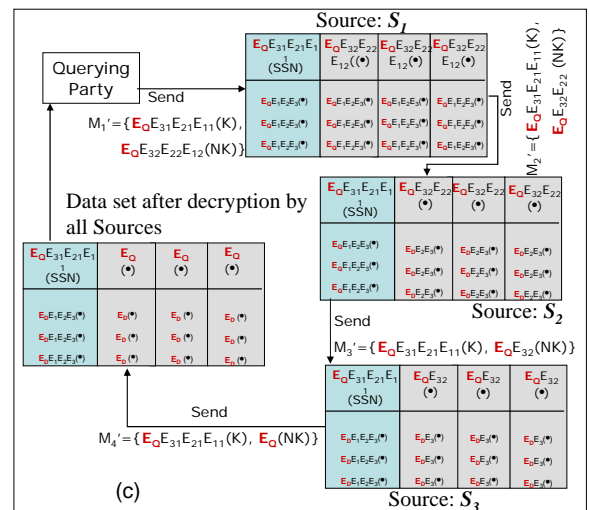
Figure 1(b) depicts the process of joining of encrypted data from three sources S_1 , S_2 , and S_3 and forwarding of the integrated dataset with all encrypted values to the querying party.



(a)



(b)



(c)

Figure 1. Graphical illustration of: (a) data flow for encryption of Prescription Drug Table with three sources; (b) joining of encrypted tables by S_1 which is sent to the querying party; (c) data flow for decryption process

Algorithm Securely computing privacy-preserving join

Require: $n > 1$ sources, each having a local relation R_i .

Require: a querying party QP , that joins the n relations. QP cannot provide any of its local relations in the joining operation.

Require: threshold $r \gg 1$, used to prevent inference of key attribute values because of too few tuples in the joined relation

```

1. {stage 1 – Hashing}
2. for all sources  $S_i$  {parallel operations} do
3.   if  $|R_i| < r$  then
4.     {the joined relation will have too few tuples
     enabling sources to infer the attribute values.
     Therefore, the join cannot be computed}
5.     broadcast ABORT return ERROR end if
6.   generate the encryption and decryption key pairs
      $(E_{i1}, D_{i1})$  and  $(E_{i2}, D_{i2})$ 
7.    $M \leftarrow \text{EncryptJoiningAttr}(R_i, E_{i1}) \cup$ 
      $\text{EncryptOtherAttr}(R_i, E_{i2})$ 
8.   Permute  $M$ 
9.   send  $M$  to source  $S_j$ , where  $j = (i+1) \bmod n$ 
10. end for
11. for each source  $S_j, j = 0 \dots n - 1$  do
12.    $M' \leftarrow$  receive from source  $S_i$ , where  $i \leftarrow (j-1)$ 
      $\bmod n$ 
13.    $M \leftarrow \text{EncryptJoiningAttr}(M', k_{j1}) \cup$ 
      $\text{EncryptOtherAttr}(M', k_{j2})$ 
14.   Permute  $M$ 
15.    $p \leftarrow (j+1) \bmod n$ 
16.   if  $S_p$  is the owner of the relation  $M''$  then
17.     send  $M''$  to  $S_1$  {or any pre-selected party
     for join computation}
18.   else
19.     send  $M''$  to  $S_p$ 
20.   end if
21. end for
22. {stage 2 – Join Computed by  $S_1$  or any pre-
     selected party}
23. for  $j = 0 \dots n - 1, j \neq i$  do
24.    $R_j^h \leftarrow$  receive from source  $S_j$ 
25. end for
26.  $R \leftarrow$  Join  $R_0^h, R_1^h, \dots, R_{n-1}^h$  on the first (joining)
     attribute of each relation
27. send  $R$  to  $QP$ 
28. {stage 3 – Decryption phase initiated by Querying
     Party}
29. if Querying Party  $QP$  then
30.   receive  $R$  from any source.
31.   generate the encryption and decryption key pair
      $(E_Q, D_Q)$ 
32.   Encrypt  $R$  with  $E_Q$ 
33.   Have each source  $S_i$  decrypt  $R$  with its
     decryption key  $D_{i2}$  one after the other finally
     sending it to  $QP$ 
34.    $QP$  decrypts the relation received,  $R$  with  $D_Q$ 
end if
35. return

```

Figure 2. Proposed Algorithm

Decryption. The joined data set forwarded to the querying party has both key and non key attributes in encrypted form. However, the querying party should be able to retrieve the values of non-key attributes without learning or inferring the actual value of the key attribute in any of the tuples. Therefore, the non-key attribute values in the joined data set need to be decrypted using the decryption keys of all sources one after the other. Since each source has its own decryption key which is not known to anyone, therefore the decryption of the non key attributes of the joined data set can only occur at each source. As stated earlier, it is sufficient for each party to decrypt these one followed by the other, to get the actual data. However, the querying party needs to ensure that none of the sources can see the real data.

Therefore, before sending the joined data set for decryption to individual sources, the querying party first encrypts all the attribute values in the data set using its encryption key E_Q . This ensures that the actual values of the non-key attributes in the joined data set can be viewed only by the querying party. After encryption, the querying party sends the joined data set to source S_1 which decrypts the non-key attribute of the joined data set using its decryption key D_{12} . Source S_1 then forwards the resulting data set to S_2 for decryption of the non key attributes using D_{22} . This process continues until all the sources have applied their decryption keys on the non-key attributes of the received data set. Finally, the data set is sent back to the querying party which applies its decryption key D_Q to retrieve the original values of the non key attributes in the joined data set. Note that during the entire process of decryption, none of the sources can learn the actual value of any of the attributes of the joined data set. Moreover after decryption, the values of the key attribute of the joined data set remain encrypted with the encryption keys of all the sources, and it is computationally infeasible for the querying party to extract the actual values of the key attribute.

The process for decryption of non-key attributes is illustrated in Figure 1(c).

Security Analysis

The algorithm depicted in Figure 2 computes the join of multiple data sets distributed among different sources without revealing any confidential/private information to the querying party. Note that the querying party cannot be a source for any data set that is integrated with other data sets for query evaluation. If the querying party is one of the sources, then it can infer the actual value of the identity (key) attribute by comparing the values of those non-key attributes that are common between the joined data set and its local

data set. For instance, consider Query 2 given in the introduction section. If this query is issued by the pharmacist who have access to the pharmacy database of prescription drugs, then the pharmacist can determine the identity of the patients infected with HIV by comparing the values of the attributes such as *Drug Name* and *Prescription Date* that are common between the joined data set (query result) and the prescription drug data set stored in the pharmacy database. Therefore, this algorithm cannot be used to evaluate queries which require joining of heterogeneously distributed data with the querying party as one of the data sources. Another limitation of the proposed algorithm is that it cannot prevent sources from inferring the attribute values if the joined data set corresponding to the query result is too small. In order to avoid this inference of attribute values, the algorithm does not compute the join if the size of any of the input data set is less than certain threshold value r .

Information security against querying party and data sources: Assuming that the querying party does not contribute any of its data set for the join operation and all parties follow the algorithm depicted in Figure 1, then the joined data set generated by the algorithm will be privacy preserving. Specifically, the querying party can view the values of non-key attributes only without learning or inferring the actual values of the key attribute in any of the tuples in the joined data set. Moreover, during the process of hashing, joining, and decryption, no source will learn the actual values of any attribute of any data set that is not owned by that source. In the following, we provide an intuitive reasoning of this claim.

Querying Party. In the joined data set, the value of the key attribute in each tuple is encrypted by all sources using their secret keys. Since, these keys are not known to the querying party, it is computationally infeasible for the querying party to learn the actual values of the key attribute from the encrypted values. Also, for any tuple in the joined data set, the value of the key attribute cannot be inferred by the querying party from other non-key attribute values as we assume that there is no data set owned by the querying party that will overlap with the joined data set in one or more attributes.

Data Sources. At any stage of the algorithm, none of the sources can extract the actual values of any of the attribute in any of the data set owned by other sources because of encryption. Moreover, the requirement that each data set must have at least r ($\gg 1$) tuples and the permutation of every data set a source receives for encryption prevent the owner of a data

set to correlate the encrypted values of any attribute with their actual values based on the ordering of tuples in the data set.

Conclusion

In this paper, we propose an approach that allows joining of heterogeneously distributed data in a secure and privacy preserving manner. The proposed approach employs a cryptography based solution, whereby the attribute values of all the qualifying data sets from every source is commutatively encrypted by all the sources using their own keys. Commutative encryption ensures that the encrypted keys from different data sets will be equal if and only if their original values are equal. The encryption prevents any source or querying party from extracting the individually identifiable or sensitive information from the joined data set. A similar commutative decryption ensures that only the querying party can extract the final result set. A future work is to address the major research challenge of integrating data without a shared complete identifier in a privacy-preserving manner.

Successful demonstration of this process could help overcome the trust barrier to having citizens share their clinical data with the NHIN for research purposes. Data owners do not need to send their raw data with a central aggregator. Further, this protocol does not require a trusted intermediary to do any joining, so potential security breaches by data aggregators can be avoided. Having a standards-based, high through-put communication channels among all joining parties in a federated system is important for this process to work efficiently.

References

1. Rindfleisch T C. Privacy, information, technology, and health care. *Communications of the ACM*. 1997;93-100
2. Health Services Research and the HIPAA Privacy Rule. <http://privacyruleandresearch.nih.gov/healthservicesprivacy.asp>.
3. Kelman C, Bass A, and Holman C. Research use of linked health data – a best practice protocol. *Australian and New Zealand J. of Public Health*. 2002;26:251-255.
4. Agarwal R, Evmfimievski A, and Srikant R. Information sharing across private databases. *ACM SIGMOD*, 2003;86-97.
5. O’Keefe C M, Yung M, Gu L, and Baxter R. Privacy – preserving data linkage protocols. *ACM WPES*. 2004;94-102.
6. Vaidya J. and Clifton C. Secure set intersection cardinality with application to association rule mining. *J. of Computer Security*, 2005;13: 593 - 622.
7. Malin B, Sweeney L. A secure protocol to distribute unlinkable health data. *AMIA Symp*. 2005:485- 489.