POSTER: PriReMat: A Distributed Tool for Privacy Preserving Record Linking in Healthcare

Diptendu Mohan Kar
Colorado State University
Diptendu.Kar@colostate.edu

Ibrahim Lazrig
Colorado State University
Ibrahim.Lazrig@colostate.edu

Indrajit Ray
Colorado State University
Indrajit.Ray@colostate.edu

Indrakshi Ray
Colorado State University
Indrakshi.Ray@colostate.edu

ABSTRACT

Medical institutions must comply with various federal and state policies when they share sensitive medical data with others. Traditionally, such sharing is performed by sanitizing the identifying information from individual records. However, such sanitization removes the ability to later link the records belonging to the same patient across multiple institutions which is essential for medical cohort discovery. Currently, human honest brokers assume stewardship of non sanitized data and manually facilitate such cohort discovery. However, this is slow and prone to error, not to mention that any compromise of the honest broker breaks the system. In this work, we describe PriReMat, a toolset that we have developed for privacy preserving record linkage. The underlying protocol is based on strong security primitives that we had presented earlier. This work describes the distributed implementation over untrusted machines and networks.

1 INTRODUCTION

PriReMat is a distributed application to perform privacy preserving record linkage in the healthcare area. The distributed application is executed by a group of healthcare providers who are ready to share (or publish) patient related data and a group of healthcare researchers (subscribers) that have the need for the data minus the personally identifying information in the data but need the ability to link records belonging to the same patient. PriReMat uses a semi-trusted third party to facilitate the record linkage. The role of the third party is to automatically and blindly perform record matching on encrypted data. The third party 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. However, PriReMat ensures that it is prevented from getting any useful information without colluding with publishers. PriReMat is based on our earlier work that is described in [6]. We also identified a security weakness in our earlier work, which we fix in PriReMat.

CCS '17, October 30-November 3, 2017, Dallas, TX, USA

ACM ISBN 978-1-4503-4946-8/17/10.
https://doi.org/10.1145/3133956.3138845

PriReMat is implemented using Oracle's Java™ technology. We used the following packages from the Java Development Kit (JDK) 1.8.0131: *java.io, java.math, java.net, java.sql, java.util.* The MySQL database management system acts as the data source. The three major components of PriReMat namely, the *Broker, Publisher, and Subscriber* are independent of each other and coordinate via passing messages. When distributed over a network, any host on the network can function as any one of the three components. The databases for the broker and publisher needs to be configured during installation. The publishers and the subscribers need to execute some protocol to know the IP address of the broker.

In this implementation, we use the El-Gamal cryptosystem along with its multiplicative homomorphic property. Although there are several APIs available for the El-Gamal cryptosystem, none of them that we could identify implement the homomorphic property. As a result, the entire cryptosystem had to be designed from scratch to include this property. We plan to release this new API to the public domain via Github so that others may benefit from our implementation.

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. In our setting, the challenge is that competing publishers who are not ready to reveal any information about their data to each other but nonetheless 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 across 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 – a requirement that we call retrospective queries.

Searchable encryption schemes where the data to be joined must be encrypted under same key (such as [2, 7, 8], cannot directly be applied to our scenario. On the other hand, private set intersection [4, 5] and multi-party computation are potential solutions but are not very efficient for large settings. Yin and Yau [9] propose a privacy preserving repository for data integration across data sharing services that allow owners to specify integration requirements and data sharing services to safeguard their privacy. However, the scheme requires a mandatory secret sharing between competing parties and is not acceptable under our setup. A similar problem occurs with the scheme proposed by Carbunar and Sion [1]. Chow et al.'s proposed protocol [3], called Two-Party Query computation,

has limited applicability to our scenario since it does not support retrospective queries.

3 PRIREMAT SCHEME CONSTRUCTION

Our scheme works in three phases: the *setup* phase, the *encryption of query results* phase, and the *secure record matching* phase. We briefly describe the phases here. The interested reader is referred to [6] for further details.

Setup: The setup phase generates the publishers' key converters that allow the broker to transform an encrypted data into another encrypted data with a different key without first decrypting it. A publisher collaborates with other publisher's to generate its key converter. When a publisher joins a group for the first time, it goes through the setup phase. An existing publisher also needs to participate in the setup phase, if a refreshing of keys is required when new publishers join the system. These key converters are delegated to the third party (broker) and are 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 be reconstructed by any of the parties, namely, individual publishers, broker and subscribers without collusion.

We use the ElGamal homomorphic cryptosystem that supports product operations 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.

Encryption of query results: 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 then redirects the data to the corresponding subscribers. After executing the query, each publisher encrypts the identifying parts of the query results using a cryptosystem that relies on the DDH (Decisional Diffie-Hellman) or DL (Discrete Logarithm) hardness assumptions, such as the ElGamal cryptosystem.

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 the scope of this work.

Secured Record Matching: 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 form each publisher 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.

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.

4 ADDRESSING SECURITY WEAKNESS OF EARLIER WORK

Our previous work [6] had a security weakness: After the completion of the setup phase, each publisher has their individual "key-converter" and "encryption key". This "key-converter" and "encryption key" does not change for each of the associated publishers unless the setup phase is executed again. This implies that if the same records are requested by the subscriber, again and again, the broker can infer this by studying the incoming encrypted records from the publisher. We eliminate this PriReMat by utilizing an additional step in the encryption of query results phase. After the completion of setup phase, when any publisher receives a query via the broker, it chooses a new random number r_{new}, encrypts it with the broker's public key and homomorphically multiplies with its existing "key-converter". Also, the publisher computes the modulo inverse of r_{new}, r_{new}^{-1} and homomorphically multiplies with the existing "encryption key". This additional step ensures that even the same record when fetched more than once will result in a different cipher text but still the common records can be determined.

5 IMPLEMENTATION

PriReMat is implemented as three independently executing components. *Broker, Publisher, and Subscriber*. In a given setup, we assume that there can be only one instance of the Broker application running. However, there can be multiple instances of the Publisher and Subscriber components. Each instance of a component executes as a multi-threaded process. Also, we assume that the databases used by the publishers and the subscribers have the same schema. PriReMat uses MySQL for these databases. Any one or all of the three components can be distributed over a network. All communications between these components are of the form of "events" and are comprised of different message types. The broker is provided a port number on which it listens for all incoming requests. A listener thread runs on that port and upon receiving a request it is passed on to another thread which processes the request depending on its type. The publisher and the subscriber need to provide the IP address and the port number of the broker to connect and register with it.

When a publisher or subscriber connects using the broker's IP address and port, a registration request is sent to the broker and upon successful registration, the broker sends a registration response with a success message and the next ID from a list of sequentially increasing ID numbers for the publishers and subscribers. The broker keeps a list of all publishers and subscribers that it is presently connected to by maintaining a database with their IP address and listening port number and ID numbers. The publisher and the subscriber when connecting chooses a randomly available port and runs a listener thread on that port. This port information is shared with the broker during registration so that the broker is able to send messages to this port. When any message is received by the publisher or subscriber on its listening port, it passes the request to another thread which then processes it depending on its type.

As soon as there are two registered publishers, the broker generates a list ("PublisherNeighborOverlay") which contains information about a publisher and its downstream publisher (next higher ID). This information is sent to each publisher at the initiation of the

setup phase. Whenever a new publisher joins the system or an existing publisher leaves the system this "PublisherNeighborOverlay" list is updated. In our implementation, the setup phase is triggered by the broker as soon as there are two registered publishers. Whenever any new publisher joins or any existing publisher leaves, the setup phase is re-invoked.

During the setup phase, the broker sends a setup initiate request to each of the active publishers and shares its cryptographic primitives - *prime, generator, and public key* and also the neighbor information. Each publisher after its designated task creates a setup forward request and forwards the result to its neighbor. The setup forward message contains an origin field and a traversed nodes field. When the setup forward message returns to the originating publisher, it compares the origin and the traversed nodes information to infer that its key-converter has been created. It then sends a setup completed message to the broker and stores the key-converter. A sequence diagram of the setup phase is described below.

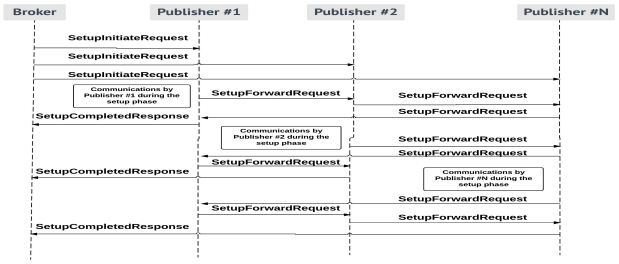

Figure 1: Setup phase sequence diagram

When the subscriber sends a query to the broker to fetch records, the broker computes the list of the publishers ("QueriedPublishersList") from which the records needs to be fetched and forwards the query to the respective publishers. The publishers after encrypting the records with their own encryption key send the result back to the broker along with their key-converter. The publishers also send an acknowledgment message to the broker notifying that it has completed sending all the records. The broker upon receiving the encrypted records from each publisher, re-encrypts them with their provided key-converter. When the broker receives all the acknowledgment messages, it compares the source with the list of all the publishers it had sent the request to. When all of the publishers have finished sending their records, the broker compares the re-encrypted records for any common record and sends the result to the subscriber. The broker keeps a mapping between the encrypted records received from each publisher and the re-encrypted record. The publishers also keep a mapping between the original record and the encrypted record. A sequence diagram of phase 2 and 3 together is described below.

When the subscriber needs to look-up more information about any record (retrospective query), it provides the re-encrypted ID to the broker. The broker from its mapping table finds the received record ID and the publisher associated with it and sends a request to those publishers. The publishers also contain their mapping table and from there each publisher finds the exact record requested and sends the requested information back to the broker which it then forwards to the subscriber.

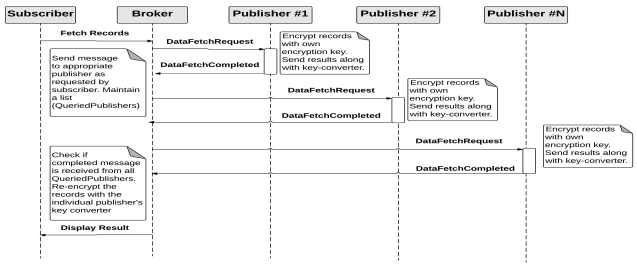

Figure 2: Phase 2 and 3 sequence diagram

6 ACKNOWLEDGEMENT

This work was partially supported by the U.S. NSF CNS under Grant No. 1650573, by the NIST under Grant No. 60NANB16D250, University of Colorado Anschutz Medical Center, CableLabs, Furuno Electric Company and SecureNok.

7 CONCLUSIONS

This work describes PriReMat, a software that we have developed for performing privacy preserving record linkage in the healthcare sector without using manually participating honest brokers. The work is based on our earlier work [6] and addresses a certain weakness in that work. PriReMat is a completely distributed application that can be ported easily to any architecture and OS supporting the Java technology. Different components of the application have been implemented as multithreaded processes. A secondary contribution of this work is the development of an API for the El-Gamal cryptosystem that enables the use of the multiplicative homomorphic property of this cryptosystem. We have tested this system on synthetic datasets and are currently working with the Anschutz Medical Center of the University of Colorado, Denver to field test it on live data.

REFERENCES

[1] B. Carbunar and R. Sion. 2012. Toward private joins on outsourced data. *Knowledge and Data Engineering, IEEE Transactions on* 24 (2012), 1699–1710.

[2] M. Chase and S Kamara. 2010. Structured encryption and controlled disclosure. In *Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security*. Singapore, 577–594.

[3] S. S. Chow, J. H. Lee, and L. Subramanian. 2009. 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.

[4] D. Dachman-Soled, T. Malkin, M. Raykova, and M. Yung. 2012. Effcient robust private set intersection. *International Journal of Applied Cryptography* 2 (2012), 289–303.

[5] S. Kamara, P. Mohassel, M. Raykova, and S. Sadeghian. 2013. *Scaling private set inter- section to billion-element sets.* Technical Report. MSR-TR-2013-63.

[6] Ibrahim Lazrig, Tarik Moataz, Indrajit Ray, Indrakshi Ray, Toan Ong, Michael G. Kahn, Frédéric Cuppens, and Nora Cuppens-Boulahia. 2015. Privacy Preserving Record Matching Using Automated Semi-trusted Broker., In Proceedings of the 29th IFIP TC 11, WG 11.3 Conference on Data and Applications Security and Privacy. *DBSec* 9149 (2015), 103–118.

[7] E. Stefanov, M. van Dijk, E. Shi, C. W. Fletcher, L. Ren, X. Yu, and S. Devadas. 2013. Path ORAM: an extremely simple oblivious RAM protocol. In *ACM Conference on Computer and Communications Security*. 299–310.

[8] M. Strizhov and I. Ray. 2014. Multi-keyword similarity search over encrypted cloud data. In *Proceedings of 29th IFIP TC 11 International Conference, Marrakech, Morocco*. 52–65.

[9] S. Yau and Y. Yin. 2008. A privacy preserving repository for data integration across data sharing services. *Services Computing, IEEE Transactions on* 1 (2008), 130–140.