Some Methods for Blindfolded Record Linkage

Author: Tim Churches and Peter Christen

Presentation by Liyuan Han
Math&Computer Science Department, Emory
Background

- The linkage of records which refer to the same entity in separate data collections is a common requirement in public health and biomedical research.
- If there is no shared common key, the partially-identifying attributes such as name, birth date, address are used.
- Someone need be given the access to match the records from different separate collections while keeping privacy protection
Related work

- Anonymous record linkage/partially-identified record linkage: the specificity and overall efficiency of the linkage operation are diminished.

- Blindfolded record linkage: the party undertaking the linkage is unable to see their actual values; good protection against a single party.
Minimal-knowledge value comparisons

- Values are character strings and scalar quantities
- Comparison methods: match exactly, similarity comparison, bigram similarity
- Cryptographically secure one-way hash functions
N-gram similarity comparators

\[ \text{Dice coefficient} = 2 \times \frac{|\text{bgrams}(x) \cap \text{bgrams}(y)|}{|\text{bgrams}(x)| + |\text{bgrams}(y)|} \]

- The 2-gram in "peter" are "pe", "et", "te" and "er"; in "pete" are "pe", "et", "te", so the bigram score is \(2 \times \frac{3}{4 + 3} = 0.86\)
Problem statement

- Alice, Bob: two separate data collections
- Carol: the third party trusted to determine match of A.a and B.a
- The actual value in A.a and B.a will not be revealed to any other parties
Protocol 1 – step 1

Alice and Bob mutually agree on:

- Secret random key, shared only A and B
- Particular keyed hash transformation function
- Standard pre-processing for strings
- A mechanism for adding “chaff” to thwart statistic attack
Protocol 1 – step 2

- Pre-process the values by computing the set of bigrams
Protocol 1 – step 3

- Compute the power sets of each set of bigrams

(), ("pe"), ("et"), ("te"), ("er"),
("pe","et"), ("pe","te"), ("pe","er"), ("et","te"),
("et","er"), ("te","er"),
("pe","et","te"), ("pe","et","er"),
("pe","te","er"), ("et","te","er"),
("pe","et","te","er")
Protocol 1 – step 4

- Discard the empty set
- Each set in the power sets is sorted to store in an attribute called `a_bigram_combination_digest`
Protocol 1 – step 5

- Record_key to indentify the records only to herself
- A_bigram_combination_length
- A_length: the counts of bigrams in each value
Protocol 1- step 6

- Alice then creates a set of tuples of the form (A.record_key_digest, A.a_bigram_combination_digest, A.a_bigram_combination_length, A.a_length), encrypts this set using Carol's public key, and sends the result to Carol. An example of tuples created for the value "peter" is given in Table 1.
Protocol 1- step 7

- Bob follows the step 2-6 as Alice
- Table 2
Protocol 1- step 8

- Carol determines the intersection of A.a_bigram_combination_digest and B.a_bigram_combination_digest.
Protocol 1 – step 9

- Carol reports the similarity scores
Security weakness

- Frequency analysis attack by Carol by the access to the number of bigrams – “chaffing”
- Alice collude with Carol or vice versa – employ multiple independent instances of Carol and decide to select one at the very last moment
Protocol 2

- The fourth party – David
- Step 1-4 are identical to Protocol 1
- Step 5: Alice prepares a set of tuples comprising 
  (A.a_bigram_combination_digest, 
  \{(Rrand, A.record_key_digest, 
  A.a_bigram_combination_length, 
  A.a_length)\}) PublicKeyD), where Rrand is an 
  arbitrary random number and {...} 
  PublicKeyD denotes that the contents are 
  encrypted with David's public key.
Protocol 2

- **Step 6**: Alice encrypts this set of tuples with Carol's public key and sends the result to her.
- **Step 7**: Bob does the same process.
- **Step 8**: Carol determines the intersection of the two attributes of keyed hash digest values. Carol then sends David a set of tuples which correspond to the intersecting bigram combination keyed hash digest values, each tuple comprised of \(\{(R_{rand}, A.\text{record_key_digest}, A.\text{a_bigram_combination_length}, A.a_{length})\text{PublicKeyD}, \{(R_{rand}, B.\text{record_key_digest}, B.\text{a_bigram_combination_length}, B.a_{length})\text{PublicKeyD}\)).
- **Step 9**: Using his private key, David decrypts the bigram length values and calculates a bigram score for tuples he has been sent. The value \(R_{rand}\) is ignored. David then determines the maximum bigram score for each unique pairing of \(A.\text{record_key_digest}\) and \(B.\text{record_key_digest}\).
Comparison of scalar values

- Treat as a sequence of bytes and transform by a keyed hash algorithm in the same way as a character string
Protocol 3

- Two new parties
- Edith: combine the results of minimal-knowledge comparisons for individual data items
- Freddy: to receive linked but de-identified records from Alice's and Bob's data collections.
Protocol 3 – step 1

- Alice and Bob dispatch the information needed for the similarity comparison task defined in Protocol 2 to different instances of Carol, which we will denote as Carola, Carolb and so on. A different shared secret hashing key. Each instance of Carol forwards the necessary information in encrypted form to the designated instance of David, who computes the similarity score (bigram_score) for each pair of Alice's and Bob's records.
Protocol 3 – step 2

- Each instance of David then sends the results of his computations to Edith as a set of tuples of the form (A.record_key_digest, B.record_key_digest, data_item_identifier, similarity_score), where data_item_identifier is the name or some other identifier of the attributes a,b,.../present in both data collections, A and B.

- Note that not every possible combination of A.record_key_digest and B.record_key_digest will be present in the data sent to Edith by each instance of David -only those record pairs for which the similarity score was greater than some threshold score.
Edith then performs an outer join of these data using the tuple (A.record_key_digest, B.record_key_digest) as the join key, to form tuples of the form:

(A.record_key_digest, B.record_key_digest, a.similarity_score, b.similarity_score, ..., i.similarity_score)
Protocol 3 – step 4

- Edith now multiplies this matrix by vectors of individual comparison weights for various degrees of similarity, and arrives at pairs of (A.record_key_digest, B.record_key_digest).
Protocol 3 – step 5

- Alice assembles the data required by Freddy for every record in her database, and encrypts each record with Freddy's public key, and associates the resulting cipher text with the corresponding record_key_digest. Alice sends these data to Edith. Bob does the same. Edith now performs an inner join of these data with her list of matching records, using the record_key_digest values as the join key. Edith forwards the encrypted data records from this inner join to Freddy, who decrypts the data for each record using his private key.
Conclusion

- Traditional methods of record linkage are vulnerable to misdeeds by or compromise of single parties, particularly the party undertaking the linkage. Although the protocols described in this paper are not unconditionally secure, they do make it much harder for any one party involved in a linkage operation to determine any useful information about the data being linked.
Thank you!

Q&A