Some methods for blindfolded record linkage

Tim Churches[1]*, Peter Christen[2]

[1] Centre for Epidemiology and Research, Population Health Division, New South Wales Department of Health, Locked Mail Bag 961, North Sydney NSW 2059, Australia

[2] Department of Computer Science, Australian National University, Canberra ACT 0200, Australia

Appendix 1 – Proof-of-concept implementation of Protocol 2

A proof-of-concept implementation of Protocol 2 is available in the form of a series of Python computer programs and two associated source data files, representing Alice's and Bob's original data. Some aspects of the protocol have been simplified for the purposes of illustration. In particular, public key encryption is not used – instead, a symmetrical encryption protocol is used. It should be noted that the symmetrical encryption protocol chosen is known to be insecure and was selected merely for illustrative purposes and to avoid problems with export controls. In practice, a known secure protocol should be used. In addition, message transfer protocols are not used – rather, each party has their own subdirectory, and the transfer of data between parties is simulated by simply writing files in the directories of other parties. However, data is only ever read from the each party's own directory. The only exception to this is an additional party which plays the rôle of an omniscient deity. This allows the bigram scores obtained by the minimum-knowledge protocol to be compared in an additional step with bigram scores calculated in the normal manner (that is, with full knowledge of the original character strings).

Interested readers may download the programmes in Additional Files 3 and 4, for Posix (Unix, Linux or Apple Mac OS X) and Microsoft Windows platforms respectively. The archive files should be unpacked in a temporary directory. This will create the required subdirectories for each entity. Each programme should be run in turn from the command line. The sequence of commands (in bold) and the resulting output (with some elision in the interests of space) are shown below. Version 2.3 or later of the Python language is also required. This may be downloaded at no cost from http://www.python.org.

Readers should note that the provided programmes are for illustrative purposes only and should not be used to process confidential data. No warranty that the programme code is fit for any particular purpose is implied or given. At the time of writing, work is under way on production-quality implementations of the protocols, to appear as part of the free, open source *Febrl* package, available at http://datamining.anu.edu.au/linkage.html.

The Python programme code and the output it produces are shown below. Comments in the Python programmes, which appear on lines starting with a hash (#) symbol, explain the programme flow.

Step 1 (Alice): Programme

```
# make required programme libraries available
import sha, random, sys, os
#
# Create the shared secret random key
K_AB = sha.new(str(random.randint(0,sys.maxint-1))).hexdigest()
print
print "Shared secret random hash key K_AB: " + K_AB
print
#
# Save the key in Alice's directory
K_AB_file_alice = open("./alice/protocol_2_K_AB.txt", "w")
K_AB_file_alice.write(K_AB)
K_AB_file_alice.close()
#
# Save the key in Bob's directory
K_AB_file_bob = open("./bob/protocol_2_K_AB.txt", "w")
K_AB_file_bob.write(K_AB)
K_AB_file_bob.close()
#
# Create a secret key shared with David (but not Carol) in lieu of
# public key cryptography to hide values from Carol
K_AD = sha.new(str(random.randint(0,sys.maxint-1))).hexdigest()
print
print "Shared secret key K_AD: " + K_AD
print
#
# Save the key in Alice's directory
K_AD_file_alice = open("./alice/protocol_2_K_AD.txt", "w")
K_AD_file_alice.write(K_AD)
K_AD_file_alice.close()
#
# Save the key in David's directory
K_AD_file_david = open("./david/protocol_2_K_AD.txt", "w")
K_AD_file_david.write(K_AD)
K_AD_file_david.close()
#
# Create a non-shared secret key
K_A = sha.new(str(random.randint(0,sys.maxint-1))).hexdigest()
print
print "Non-shared secret key K_A: " + K_A
print
#
# Save the key in Alice's directory
K_A_file_alice = open("./alice/protocol_2_K_A.txt", "w")
K_A_file_alice.write(K_A)
K_A_file_alice.close()
# -----------------------------------------------------------------
```

Step 1 (Alice): Output

```
$ python p2_s1_alice.py
Shared secret random hash key K_AB: 07a7a77e53d1ad8a92dbd0457b3c2636a242aeaa
Shared secret key K_AD: 27bde971ac3f5017dabdd8ab5f85ba4c0cd6bd5a
Non-shared secret key K_A: 77529ee1ba5c12859a95412a9cec148f05e50cd3
```

Steps 2-6 (Alice): Programme

```
# Turn off warning about insecurity of rotor encryption
import warnings
warnings.filterwarnings("ignore","",DeprecationWarning)
# make required programme libraries available
import sha, hmac, os, rotor, random, pickle, sys

# Note: For illustrative purposes only, this program uses the Python
# rotor symmetrical encryption module (which is known to be insecure)
# and a shared secret key instead of public key encryption as specified
# prescribed in Protocol 2
def enigmatise(objects,key):
  plaintext = pickle.dumps(objects)
  rt = rotor.newrotor(key)
  ciphertext = rt.encrypt(plaintext)
  return ciphertext

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Recursive function to calculate sub-list permutations
def sublist_comb(in_list, length):
  """Routine to recursively compute all combinations of sub-lists
     of the given 'in_list' of length 'len'.
     Based on Gagan Saksena's routines from Activestate Python
     cookbook, see:
     http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/66465
     and modified by Ole Nielsen, MSI ANU, November 2002
  """
  sub_lists = []
  in_list_len = len(in_list)  # Number of elements in in_list
  if (length == 0):
    return [[]]
  else:
    for i in range(in_list_len):
      sub = sublist_comb(in_list[i+1:], length-1)
      for l in sub:
        l.insert(0, in_list[i])
      sub_lists += sub
  return sub_lists
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

# Read the non-shared secret key K_A written by Alice in step 1
K_A_file = open("./alice/protocol_2_K_A.txt", "r")
K_A = K_A_file.readline()
K_A_file.close()

# Read the shared secret key K_AB written by Alice in step 1
K_AB_file = open("./alice/protocol_2_K_AB.txt", "r")
K_AB = K_AB_file.readline()
K_AB_file.close()

# Read the shared secret key K_AD written by Alice in step 1
K_AD_file = open("./alice/protocol_2_K_AD.txt", "r")
K_AD = K_AD_file.readline()
K_AD_file.close()

# Load and then print original database A.a
A_a_file = open("./alice/A_database.txt", "r")
A_original_records = A_a_file.readlines()
```

```
A_a_file.close()

print
print "Database A original records:"
for r in A_original_records:
  print " ", r.strip()
print

# Empty lists which will hold the transformed and encoded data
A_hash_bigram_comb =     []
A_hash_bigram_comb_len = []
A_encrypt_rec_key =      []
A_len =                  []

# List to hold original data and record keys for use by deity
A_source_database = []

rec_id = 0  # Counter for the record identifiers

# Iterate through Alice's original records
for r in A_original_records:
  str_val = r.strip().lower()
  str_val_len = len(str_val)

  # Number of bigrams in string
  str_bigram_num = str_val_len - 1

  # Create the bigram list from the original record value
  bigram_list = []

  # enumerate all the bigrams in the string
  for i in range(1, str_val_len):
    bigram = str_val[i-1:i+1]
    if bigram not in bigram_list:
        bigram_list.append(bigram)
  # sort the resulting list of bigrams
  bigram_list.sort()

  # Count of bigrams in the list
  num_bigrams = len(bigram_list)

  # Form all combinations of bigrams with sub-lists of various lengths
  # i.e. the powerset(sans the null set)
  for sub_list_len in range(1,num_bigrams+1):
    bigram_comb_list = sublist_comb(bigram_list, sub_list_len)
    # Now process each combination in the powerset
    for bigram_comb in bigram_comb_list:
      # concatenate the sorted bigrams into a string
      bigram_comb_str = ''.join(bigram_comb)
      # calculate the keyed hash for the bigram combination string
      encoded_value = hmac.new(K_AB,bigram_comb_str,sha).hexdigest()
      # Append information about the combination to lists for output
      A_hash_bigram_comb.append(encoded_value)
      A_hash_bigram_comb_len.append(sub_list_len)
      A_len.append(str_bigram_num)
      A_encrypt_rec_key.append(hmac.new(K_A,str(rec_id),sha).hexdigest())
  # Print original and encrypted record keys
  print "Record key: %i, encrypted: %s" % \
        (rec_id, A_encrypt_rec_key[-1])

  # Append these data items to A database
  A_source_database.append((rec_id, A_encrypt_rec_key[-1],str_val))
  # increment record counter
  rec_id += 1

# Save the tuples in serialised form in Carol's directory
tuples_A = []
for i in range(0, len(A_hash_bigram_comb)):
  R = sha.new(str(random.randint(0,sys.maxint-1))).hexdigest() # Random payload
  tuple_A = (A_hash_bigram_comb[i],enigmatise((A_encrypt_rec_key[i],\
                                           A_hash_bigram_comb_len[i],\
                                           A_len[i],R),K_AD))
  tuples_A.append(tuple_A)
# Use Python "pickles" to serialise the data for "sending" to Carol
tuples_A_file_carol = open("./carol/protocol_2_tuples_A.pickle", "w")
pickle.dump(tuples_A,tuples_A_file_carol)
tuples_A_file_carol.close()

# print output
print
print "Database A processed, bigram hashes and encrypted data for David:"
print " Hashed bigram combinations"
for tuple_A in tuples_A:
  print " ", tuple_A[0]
print

print "Created %i encrypted records from %i original records" % \
      (len(A_hash_bigram_comb), rec_id)

# Save the original records and encrypted key in deity's
# directory for later comparison - this is only for illustrative
# purposes, of course.
A_original_records_file_deity = open
("./deity/protocol_2_A_original_records.pickle", "w")
pickle.dump(A_source_database,A_original_records_file_deity)
A_original_records_file_deity.close()
# ----------------------------------------------------------------
```

Step 2-6 (Alice): Output

```
$ python p2_s2-6_alice.py
Database A original records:
  Shackleford
  Dunningham
  Nichleson
  Jones
  Massey
  Abroms
  Hardin
  Itman
  Jeraldine
  Marhta
  Michelle
  Julies
```

```
    Tanya
    Dwayne
    Sean
    Jon
    Brookhaven
    Brook Hallow
    Decatur
    Higbee
    Lacura
    Iowa
    1st

Record key: 0, encrypted: 2f4628821ed2cc4c02ff3a0f8b03357bb029ab64
Record key: 1, encrypted: aa32dc1a4c9e30469d0196ea4f9e4e502c97e2e7
Record key: 2, encrypted: 3762d8af035bc225c7b19782d81c352acc5d4f9e
⋮
Record key: 21, encrypted: ddab44b583df7694abf645db4771ec13011ae3ff
Record key: 22, encrypted: 1053c1109005781c06c5b7ca12954bc0b1499e2b
Record key: 23, encrypted: cc40989298fd625e2134e8d98f54e776e21379d3

Database A processed, bigram hashes and encrypted data for David:
  Hashed bigram combinations
    7dada3572c820d5f1004b3f5f34a7f878aa53618
    6a93337481ab2f2f5341d8d171af3dc739396e84
    a8ea363c03752a4658cacedc9c1bb9e2b8682cc3
    ⋮
    bc94b23786077ba3061e46963da47b6cebd69a5d
    6dcad0163f504504898aa4a1f1ccef182b6db297
    5d2f9ea669da70601f43c3d13b07f5d4465fc238

Created 6128 encrypted records from 23 original records
```

Step 7 (Bob): Programme

This is essentially the same as Steps 2-6 above, except from Bob's perspective, and has been ommitted in the interests of brevity. The programme code is available in Additional Files 1 and 2.

Step 7 (Bob): Output

$ python p2_s7_bob.py

```
Non-shared secret key K_B: d1a4011e6ce8d5f034d6e6d638c2e0c7e3dca728
Shared secret key K_BD: ecb4fcfb426551374c51ed91756890ced911918b

Database B original records:
    Shackelford
    Cunnigham
    Nichulson
    Johnson
    Massie
    Abrams
    Martinez
    Smith
    Geraldine
    Martha
    Michael
    Julius
    Tonya
    Duane
    Susan
    John
    Jan
    Brrokhaven
    Brook Hllw
    Decatir
    Highee
    Higvee
    Locura
    Iona
    Ist

Record key: 0, encrypted: 252eeec1606490f5c6ec5cad8ed8d25d9c1cfd84
Record key: 1, encrypted: 0ebac3d3d9a3b504e9e8a28082f682e21829ea52
Record key: 2, encrypted: f7bb5a26ce4e0b70cc06af866cf167f749952f57
⋮
Record key: 23, encrypted: e134487b085f29bb20a7bc65fd6c7f347f926483
Record key: 24, encrypted: babe5569e996d55d6de0451ab1c753343ea79799
Record key: 25, encrypted: b03deb419dbeb9070ac271c863dc1ccf50467383

Database B processed, bigram hashes and encrypted data for David:
  Hashed bigram combinations
    7dada3572c820d5f1004b3f5f34a7f878aa53618
    6a93337481ab2f2f5341d8d171af3dc739396e84
    b9a332e20bcde3f6db9949dbf55a20021768a6a7
    ⋮
    ec53fc4a29ca03b5cb20b03588c5de40fee4c6fc
    6dcad0163f504504898aa4a1f1ccef182b6db297
    9cbee84f883d4bf2638b282278bc40b5224a0964

Created 4446 encrypted records from 25 original records
```

Step 8 (Carol): Programme

```
import pickle

# Read the hashed tuples received from Alice
tuples_A_file_carol = open("./carol/protocol_2_tuples_A.pickle", "r")
tuples_A = pickle.load(tuples_A_file_carol)
tuples_A_file_carol.close()
# And print them out for reference
print
print "Alice's hashed bigrams and other data (encrypted with David's key) loaded"
print
# Read the hashed tuples received from Bob
tuples_B_file_carol = open("./carol/protocol_2_tuples_B.pickle", "r")
tuples_B = pickle.load(tuples_B_file_carol)
tuples_B_file_carol.close()
# And print them out for reference
print
print "Bob's hashed bigrams and other data (encrypted with David's key) loaded"
print

# Create two dictionaries with hashed bigrams as keys
database_A = {}
for tup in tuples_A:
  hash_bigr_comb = tup[0]
  encrypted_data_for_David = tup[1]
  if database_A.has_key(hash_bigr_comb):
    database_A[hash_bigr_comb].append(encrypted_data_for_David)
  else:
    database_A[hash_bigr_comb] = [encrypted_data_for_David]
database_B = {}
for tup in tuples_B:
  hash_bigr_comb = tup[0]
  encrypted_data_for_David = tup[1]
  if database_B.has_key(hash_bigr_comb):
    database_B[hash_bigr_comb].append(encrypted_data_for_David)
  else:
    database_B[hash_bigr_comb] = [encrypted_data_for_David]

# Find intersection of records in database_A and database_B
print "Finding intersecting bigram hash digests"
print
tuples_for_David = []
for bigram_hash in database_A.keys():
  A_encrypted_data = database_A[bigram_hash]
  B_encrypted_data = database_B.get(bigram_hash,[])
  for B_data in B_encrypted_data:
    for A_data in A_encrypted_data:
      tuples_for_David.append((A_data,B_data))

print "Number of intersecting bigram hash digests:", len(tuples_for_David)
print

# Use Python "pickles" to serialise the data for "sending" to David
tuples_file_david = open("./david/protocol_2_tuples_C.pickle", "w")
pickle.dump(tuples_for_David,tuples_file_david)
tuples_file_david.close()
print "Tuples for David written to his directory"
print
# -----------------------------------------------------------------
```

Step 8 (Carol): Output

```
$ python p2_s8_carol.py
Alice's hashed bigrams and other data (encrypted with David's key) loaded
Bob's hashed bigrams and other data (encrypted with David's key) loaded
Finding intersecting bigram hash digests...
Number of intersecting bigram hash digests: 1091
Tuples for David written to his directory
```

Step 9 (David): Programme

```
# Turn off warning about rotor encryption
import warnings
warnings.filterwarnings("ignore","",DeprecationWarning)
# make required libraries available
import pickle, rotor

# Note: For illustrative purposes only, this program uses the Python
# rotor symmetrical encryption module (which is known to be insecure)
# and a shared secret key instead of public key encryption as specified
# prescribed in Protocol 2
def de_enigmatise(ciphertext,key):
  rt = rotor.newrotor(key)
  plaintext = rt.decrypt(ciphertext)
  objects = pickle.loads(plaintext)
  return objects

# Read the shared secret key from Alice
print "reading shared secret keys from Alice and Bob..."
print
K_AD_file_david = open("./david/protocol_2_K_AD.txt", "r")
K_AD = K_AD_file_david.readline()
K_AD_file_david.close()

# Read the shared secret key from Bob
K_BD_file_david = open("./david/protocol_2_K_BD.txt", "r")
K_BD = K_BD_file_david.readline()
K_BD_file_david.close()

# Read the tuples received from Carol
print "Reading tuples received from Carol..."
print
tuples_file_david = open("./david/protocol_2_tuples_C.pickle", "r")
tuples_from_Carol = pickle.load(tuples_file_david)
tuples_file_david.close()

# Process each tuple and calculate the bigram score, storing the
# results in a diction with (A.record_key_has, B.record_key_hash)
# as the key
print "Decrypting and calculating bigram score for each tuple..."
print
bigram_scores_dict = {}
for tup in tuples_from_Carol:
  A_tuple = de_enigmatise(tup[0],K_AD)
  B_tuple = de_enigmatise(tup[1],K_BD)
  # Check that bigram lengths are the same
  if A_tuple[1] != B_tuple[1]:
    print "Error: bigram lengths not equal!"
  bigram_score = A_tuple[1] / (0.5 * (A_tuple[2] + B_tuple[2]))
  if bigram_scores_dict.has_key((A_tuple[0],B_tuple[0])):
    bigram_scores_dict[(A_tuple[0],B_tuple[0])].append(bigram_score)
  else:
    bigram_scores_dict[(A_tuple[0],B_tuple[0])] = [bigram_score]

# Now create a dictionary of maximum bigram score for each pair of records
# where the score is 0.5 or more
D_max_scores = {}
for k in bigram_scores_dict.keys():
  max_score = max(bigram_scores_dict[k])
  if max_score >= 0.5:
    D_max_scores[k] = max_score

# Send these results to the deity for checking
print "Writing bigram score results to deity's directory..."
print

results_file_deity = open("./deity/protocol_2_results_D.pickle", "w")
pickle.dump(D_max_scores,results_file_deity)
results_file_deity.close()
# ----------------------------------------------------------------
```

Step 9 (David): Output

```
$ python p2_s9_david.py
Reading shared secret keys from Alice and Bob...
Reading tuples received from Carol...
Decrypting and calculating bigram score for each tuple...
Writing bigram score results to deity's directory...
```

Step 10 (deity): Programme

```
# make libraries available
import pprint, pickle, sys
# unserialise the A database
A_original_records_file_deity = open
("./deity/protocol_2_A_original_records.pickle", "r")
A_source_database = pickle.load(A_original_records_file_deity)
A_original_records_file_deity.close()
# unserialise the B database
B_original_records_file_deity = open
("./deity/protocol_2_B_original_records.pickle", "r")
B_source_database = pickle.load(B_original_records_file_deity)
B_original_records_file_deity.close()
# unserialise the blindfolded scores from David
B_original_records_file_deity = open
("./deity/protocol_2_B_original_records.pickle", "r")
B_source_database = pickle.load(B_original_records_file_deity)
B_original_records_file_deity.close()
# unserialise the results from David
results_file_deity = open("./deity/protocol_2_results_D.pickle", "r")
D_max_scores = pickle.load(results_file_deity)
results_file_deity.close()

# Now calculate the scores from the plaintext original values
scores = {}
for A_source_record in A_source_database:
  for B_source_record in B_source_database:
    A_string = A_source_record[2]
    B_string = B_source_record[2]
    A_reckey_hash = A_source_record[1]
    B_reckey_hash = B_source_record[1]
    A_reckey = A_source_record[0]
    B_reckey = B_source_record[0]
    bigr1 = []
    bigr2 = []
    str1 = A_string
    str2 = B_string
    # Make a list of bigrams for both strings
    for i in range(1,len(str1)):
      bg = str1[i-1:i+1]
      if bg not in bigr1:
        bigr1.append(bg)
    for i in range(1,len(str2)):
      bg = str2[i-1:i+1]
      if bg not in bigr2:
        bigr2.append(bg)
    # Compute average number of bigrams
    average = (len(bigr1)+len(bigr2)) / 2.0
    if (average == 0.0):
      w =  0.0
    else:
      # Determine which bigrams are in common
      common = 0.0
      if (len(bigr1) < len(bigr2)):  # Count using the shorter bigram list
        short_bigr = bigr1
        long_bigr  = bigr2
      else:
        short_bigr = bigr2
        long_bigr  = bigr1
      for b in short_bigr:
        if (b in long_bigr):
          common += 1.0
      w = common / average
    nonsecret_bg_score = w
    if nonsecret_bg_score >= 0.5:
      scores[(A_reckey_hash, B_reckey_hash)] = \
            (nonsecret_bg_score, A_reckey, B_reckey, A_string, B_string)

print "%15s %15s     %15s     %15s" % ("A_string", "B_string", \
"Deity_bigram_score", "Blind_bigram_score")

for s in scores.keys():
  nonsecret_bg_score, A_reckey, B_reckey, A_string, B_string = scores[s]
  blindfolded_bg_score = D_max_scores[s]
  print "%15s %15s     %3f               %3f" % (A_string, B_string, \
nonsecret_bg_score, blindfolded_bg_score)
# -----------------------------------------------------------------
```

Step 10 (deity): Output

```
$ python p2_s10_deity.py
```

```
       A_string        B_string     Deity_bigram_score     Blind_bigram_score
      jeraldine       geraldine     0.875000               0.875000
          tanya           tonya     0.500000               0.500000
         higbee          highee     0.600000               0.600000
         massey          massie     0.600000               0.600000
     dunningham       cunnigham     0.705882               0.705882
         higbee          higvee     0.600000               0.600000
         abroms          abrams     0.600000               0.600000
         lacura          locura     0.600000               0.600000
      nichleson       nichulson     0.625000               0.625000
            1st             ist     0.500000               0.500000
       michelle         michael     0.615385               0.615385
            jon         johnson     0.500000               0.500000
     brookhaven      brrokhaven     0.888889               0.888889
   brook hallow      brook hllw     0.700000               0.700000
    shackleford     shackelford     0.700000               0.700000
        decatur         decatir     0.666667               0.666667
         julies          julius     0.600000               0.600000
```