**Linkja-Matching**

To begin, on cmd, cd to directory containing linkja matching jar file and linkjacrypto.dll (for matching)

**Modes**

There are 4 modes available with linkja matching: decrypt, load, match & report data

Key arguments

-jar <arg> Path to jar file

--directory <arg> Path to project root directory

Below are mode specific arguments

**DECRYPT:** It looks inside specified root directory’s > data > input to load files with matching suffix (.txt or .csv) & prefix on to sqlite database

java -Djava.library.path=. -jar Matching-1.2.0-beta-jar-with-dependencies.jar --decrypt --directory "P:\linkja-matching-master\project2" --prefix hashes --suffix .enc --decryptionKey private-agg.key

--decrypt activate decryption mode

--suffix <arg> suffix of files that should be decrypted

--prefix <arg> file extension of files that should be decrypted (.enc)

--decryptionKey <arg> path and filename of the RSA 2048 bit private key

**LOAD DATA**: It looks inside specified root directory’s > data > input to load files with matching suffix (.txt or .csv) & prefix on to sqlite database

java -jar Matching-1.2.0-beta-jar-with-dependencies.jar --directory "P:\linkja-matching-master\project2" --load --prefix hashes --suffix .csv

--load activate mode to load data files to database

--suffix <arg> suffix of files that should be loaded to database

--prefix <arg> file extension of files that should be loaded

Every time load data script runs, it appends the data to database. To remove previous data, please use delete statements below and then click ‘write changes’

Delete from GlobalMatch;

Delete from InclusionPatients;

Delete from ExclusionPatients;

**MATCH DATA**: It matches data in the database inside root directory’s > data using the specified rule number & starting ID/seed

java -jar Matching-1.2.0-beta-jar-with-dependencies.jar --directory " P:\linkja-matching-master\project2" --match 0 --seed 1

--match <arg> provide matching rule to match data in database

--seed <arg> number between 1 and 1000, global id assignment will begin from seed+1

**REPORT DATA**: It extracts data (site id, project name, patient id & global match id) for the site and saves the resulting file in data>output folder

java -jar Matching-1.2.0-beta-jar-with-dependencies.jar --directory " P:\linkja-matching-master\project2" --report 6

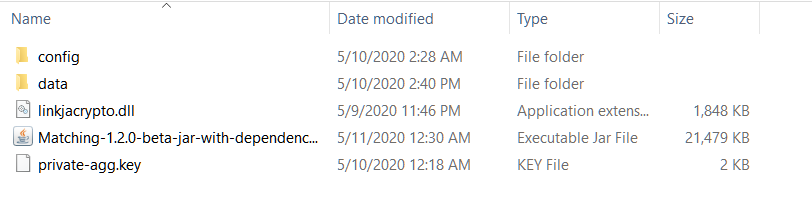
--report <arg> provide site id to extract site id & global id

**GUI:** For smaller datasets, GUI can be used to load, match & report data

java -jar Matching-1.2.0-beta-jar-with-dependencies.jar --directory " P:\linkja-matching-master\project2" --GUI

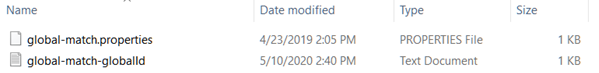
**Linkja matching folder structure**

1. Project root directory:



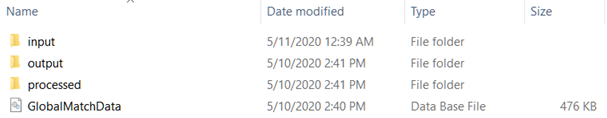
* Replace the private key file with project’s RSA 2048-bit private key
* Download latest jar file from <https://github.com/linkja/linkja-matching/releases>
* Download latest matching linkjacrypto.dll from <https://github.com/linkja/linkja-matching/tree/master/lib>

2. config:



No changes

3. data:



* ***input*** Contains all data files (encrypted files and decrypted files)
* ***output*** *Contains* distributable file with site id, project id, patient id, global id to share with the data contributors
* ***processed*** Contains log report

**Additional Utilities**

**1. OpenSSL:** Aggregator can generate RSA 2048-bit public-private key pair using openssl. Windows build is available on box <https://northwestern.box.com/s/jo0i43676clexastg5p795r0bejzaf10>

On cmd, cd to openssl.exe (.\OpenSSL\vc-win64a\bin) and to generate keys,

private key: openssl genrsa -out private-agg.key 2048

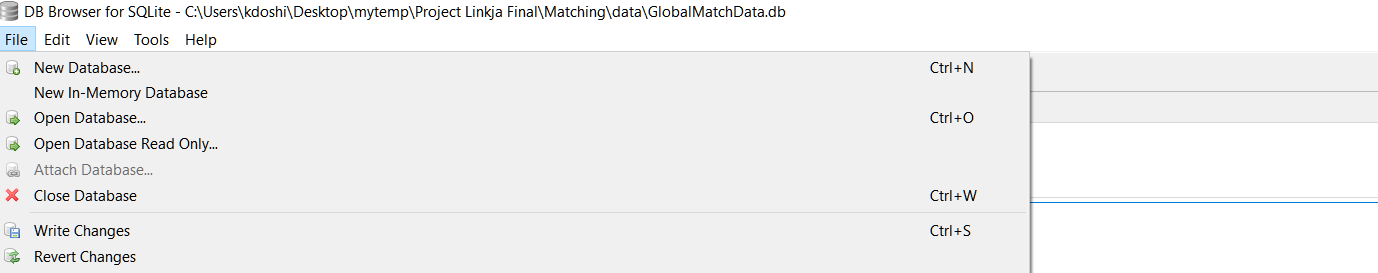
public key from the private key: openssl rsa -in private-agg.key -outform PEM -pubout -out public-agg.key

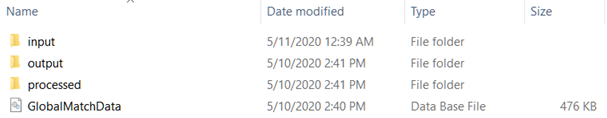
Name the files (private-agg and public-agg) as desired but keep extension .key

**2. SQLite database:** Download 64-bit DLL (x64) for SQLite version 3.31.1 from <https://www.sqlite.org/download.html>

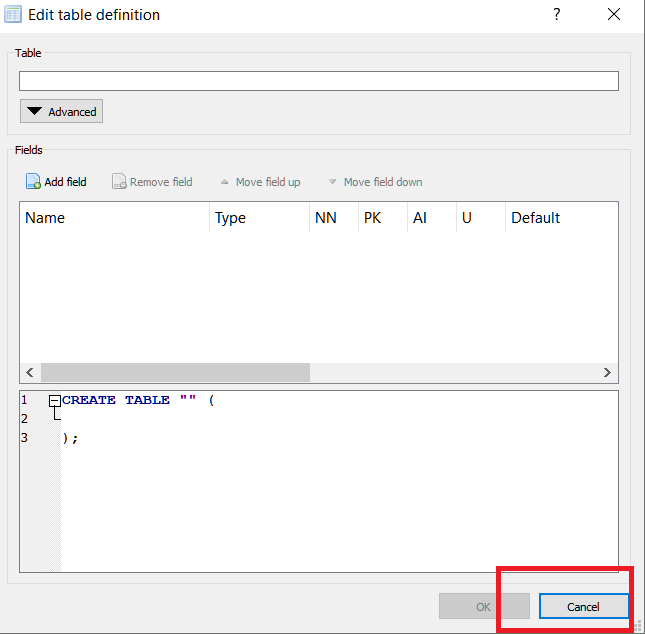
**3. SQLite browser:** Download standard installer for 64-bit Windows from <https://sqlitebrowser.org/dl/>

- To create new database or open database, click on file, navigate to your project>data folder e.g., P:\linkja-matching-master\project2\data, and create/save GlobalMatchData database

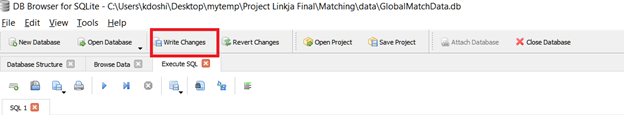




When edit table dialog opens, hit cancel and follow DDL instructions below to create tables



**4. DDL** to create tables in SQLite browser is in Appendix. Copy & paste the ddl in Execute SQL tab, and hit play/run button. Make sure to click ‘write changes’ after executing DDL statements to commit changes to database



**5. To delete data from SQLite browser**

Delete from GlobalMatch;

Delete from InclusionPatients;

Delete from ExclusionPatients;

Make sure to click ‘write changes’ after executing delete statements to commit changes to database

**6. Optional JVM memory parameters** to set starting and maximum JVM memory size:

java -Xms3G -Xmx6G

sample usage:

java -Xms3G -Xmx6G -jar Matching-1.2.0-beta-jar-with-dependencies.jar --directory "P:\linkja-matching-master\project2" --load --prefix hashes --suffix .csv

**Appendix: DDL Statements**

CREATE TABLE IF NOT EXISTS InclusionPatients (

id integer PRIMARY KEY,

globalId integer,

siteId text,

projectId text,

pidhash text NOT NULL,

hash1 text,

hash2 text,

hash3 text,

hash4 text,

hash5 text,

hash6 text,

hash7 text,

hash8 text,

hash9 text,

hash10 text

);

CREATE TABLE IF NOT EXISTS ExclusionPatients (

id integer PRIMARY KEY,

globalId integer,

siteId text,

projectId text,

pidhash text NOT NULL,

hash1 text,

hash2 text,

hash3 text,

hash4 text,

hash5 text,

hash6 text,

hash7 text,

hash8 text,

hash9 text,

hash10 text

);

CREATE TABLE IF NOT EXISTS GlobalMatch (

id integer PRIMARY KEY,

globalId integer,

siteId text,

projectId text,

pidhash text NOT NULL,

hash1 text,

hash2 text,

hash3 text,

hash4 text,

hash5 text,

hash6 text,

hash7 text,

hash8 text,

hash9 text,

hash10 text

);

CREATE INDEX match0 ON GlobalMatch (hash1,hash2,hash3,hash4,hash5,hash6,hash7,hash8,hash9,hash10,id);

CREATE INDEX match1 ON GlobalMatch (hash1,hash2,hash5,hash9,hash10,id);

CREATE INDEX match2 ON GlobalMatch (hash3,hash4,hash6,id);

CREATE INDEX match3 ON GlobalMatch (hash1,id);

CREATE INDEX match4 ON GlobalMatch (hash1,hash2,id);

CREATE INDEX match5 ON GlobalMatch (hash1,hash5,id);

CREATE INDEX match6 ON GlobalMatch (hash1,hash9,id);

CREATE INDEX match7 ON GlobalMatch (hash1,hash10,id);

CREATE INDEX match8 ON GlobalMatch (hash3,id);

CREATE INDEX match9 ON GlobalMatch (hash3,hash4,id);

CREATE INDEX match10 ON GlobalMatch (hash3,hash6,id);

CREATE INDEX match11 ON GlobalMatch (hash7,id);

CREATE INDEX match12 ON GlobalMatch (hash8,id);

CREATE INDEX pidindex ON GlobalMatch (pidhash,siteId,projectId,id);

COMMIT;

CREATE VIEW report1 AS

SELECT siteId, projectId, pidhash, globalId FROM GlobalMatch;

**Appendix: Matching Rules**

Deterministic algorithms determine whether record pairs agree or disagree on a set of identifiers, where agreement on a given identifier is assessed as a discrete—“all-or-nothing”—outcome[[1]](#footnote-1). The match occurs on a set of identifiers that have been standardized, combined and hashed similarly across all participating sites (handled in Module 2: Hashing). When using composite identifiers (e.g., First Name + Last Name + Date of Birth + Last 4 SSN), equal weight is assigned to each data element and only when the entire composite ID matches, it is flagged as a match. In our current build, 10 composite identifiers are processed that allow syllogistic matches including, iterative match, hierarchical match (from more specific to more sensitive), and fuzzy match (partial name match). Also, see Appendix C, Table 1 for full list of rules. Deterministic algorithm is simple to understand, easy to implement and quality of matches can be improved greatly by improving data normalization techniques and increasing identifiers. Below are key match rules:

Rule 0: any hash <--> any hash

Rule 1: any hash with Full Name DOB SSN <--> any hash with Full Name DOB SSN

Rule 2: any hash with Full Name DOB <--> any hash with Full Name DOB

Rule 3: Hash 1 Full Name DOB SSN <--> Hash 1 Full Name DOB SSN

Rule 4: Hash 1 Full Name DOB SSN <--> Hash 2 Transposed Full Name DOB SSN

Rule 5: Hash 1 Full Name DOB SSN <--> Hash 5 Full Name Transposed DOB SSN

Rule 6: Hash 1 Full Name DOB SSN <--> Hash 9 Full Name DOB 1 Day SSN

Rule 7: Hash 1 Full Name DOB SSN <--> Hash 10 Full Name DOB 1 Year SSN

Rule 8: Hash 3 Full Name DOB <--> Hash 3 Full Name DOB

Rule 9: Hash 3 Full Name DOB <--> Hash 4 Transposed Full Name DOB

Rule 10: Hash 3 Full Name DOB <--> Hash 6 Full Name DOB 1 Year SSN

Rule 11: Hash 7 Partial Name DOB SSN <--> Hash 7 Partial Name DOB SSN

Rule 12: Hash 8 Partial Name DOB <--> Hash 8 Partial Name DOB

Notes:

* Full Name = First Name + Last Name
* Transposed Full Name = Last Name + First Name
* DOB = Date of Birth (MM/DD/YYYY)
* Transposed DOB = Transposed Date and Month in Date of Birth (DD/MM/YYYY)
* SSN = Last 4 Social Security Numbers
* Partial Name = Initial 3 characters in First Name + Last Name
* Rules 0, 1, and 2 are legacy rules. These are available for backward compatibility and will be phased out
* Multiple rules can be combined to increase sensitivity e.g., --match 3,4,5,6

Table1: Complete syllogistic matching set rules available in Java:

|  |  |  |
| --- | --- | --- |
| **Rule No. (Match set)** | **Composite identifiers (Hashed)** | **Match Rule Description** |
| 0\* | All hashes with each other | All hashes with each other |
| 1\* | hash1,hash2,hash5,hash9,hash10 with hash1,hash2,hash5,hash9,hash10 | All Full name, dob & SSN |
| 2\* | hash3,hash4,hash6 with hash3,hash6,hash6 | All Full name & dob |
| 3 | hash1 with hash1 | Full name, dob & SSN |
| 4 | hash1 with hash2 | Full transposed name, dob & SSN |
| 5 | hash1 with hash5 | Full name, transposed dob & SSN |
| 6 | hash1 with hash9 | Full name, day diff dob & SSN |
| 7 | hash1 with hash10 | Full name, year diff dob & SSN |
| 8 | hash3 with hash3 | Full name, dob |
| 9 | hash3 with hash4 | Full transposed name, dob |
| 10 | hash3 with hash6 | Full name, transposed dob |
| 11\*\* | hash7 with hash7 | Partial name, dob & SSN\* |
| 12\*\* | hash8 with hash8 | Partial name, dob\* |

\*Rules 0,1,2 are legacy match rules carried forward for backward compatibility

\*\*Rules 11 and 12 are fuzzy matches allowing first 3 initials of first name combined with rest of the elements as specified in description

The rules can be combined to form more complex algorithms. Recommended cross match set rule combinations:

3 – most specific

3,4,5,6,7

3,4,5,6,7,11

8,9,10

12

12,9,10,6,7 – most sensitive

**Appendix: Config files**

**1. Create and save global-match.properties with below code**

############ Define Global Patient Match properties ###########

# database information

Database=SQLite

DbDirectory=%ProjectRoot%/data/

DbName=GlobalMatchData.db

# directory for configuration files

ConfigFilesDirectory=%ProjectRoot%/config

# directory for input files

InputFilesDirectory=%ProjectRoot%/data/input

# directory for CDA output files

OutputFilesDirectory=%ProjectRoot%/data/output

# directory for processed files

ProcessedFilesDirectory=%ProjectRoot%/data/processed

#Input file name prefix - suffix for hash files

InputFileNamePrefix=hash\_

InputFileNameSuffix=.dx

#Sequence of Patient Matching Rules. comma separated

MatchingRules=3

**2. Create and save global-match-globalId.txt with below code**

1 | 2020-05-10

1. https://www.ncbi.nlm.nih.gov/books/NBK253312/ [↑](#footnote-ref-1)