

# Hashing and Anonymizing Datasets

---

LEOSON HOAY



# Contents

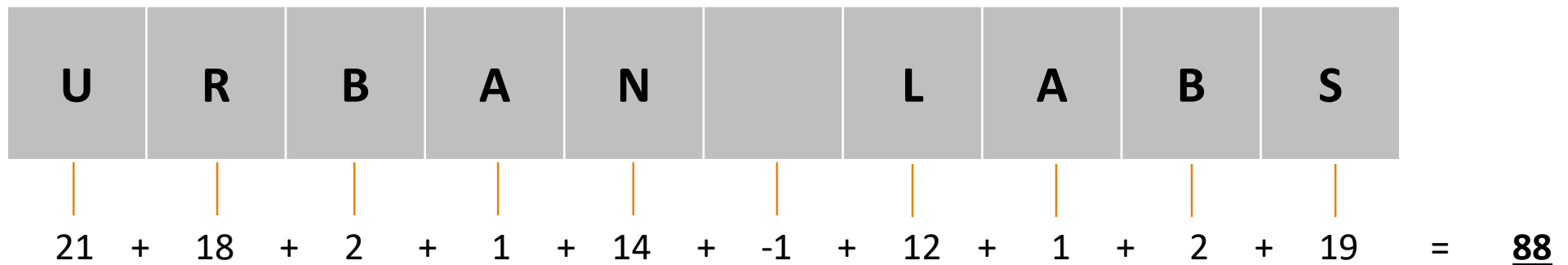
---

- What is “hashing”?
- Protocol
- Examples
- Actual Code
- Merging

# What is “hashing”?

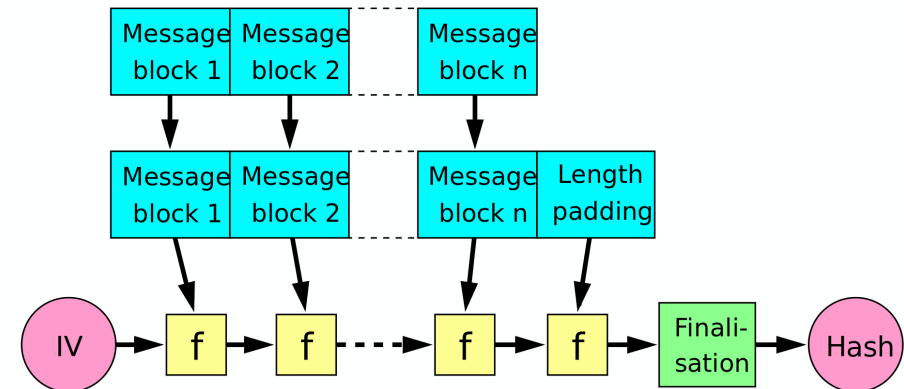
---

- A **hash** is a value that is obtained from the **transformation of a string of characters by a certain function**.
- This function can be anything – including something as simple as taking the index of each alphabet in a word and summing them up.



# What is “hashing”?

- In encryption, much more complicated functions are used to ensure:
  - **Minimal collisions** (Two different strings should not map to the same value)
  - **Difficulty of reversing the hash** without using ‘brute-force’ methods
- Some types:
  - MD5
  - SHA0
  - SHA1
  - SHA256
  - etc.



[Merkle-Damgård construction](#)

Image: Wikipedia

# What is “hashing”?

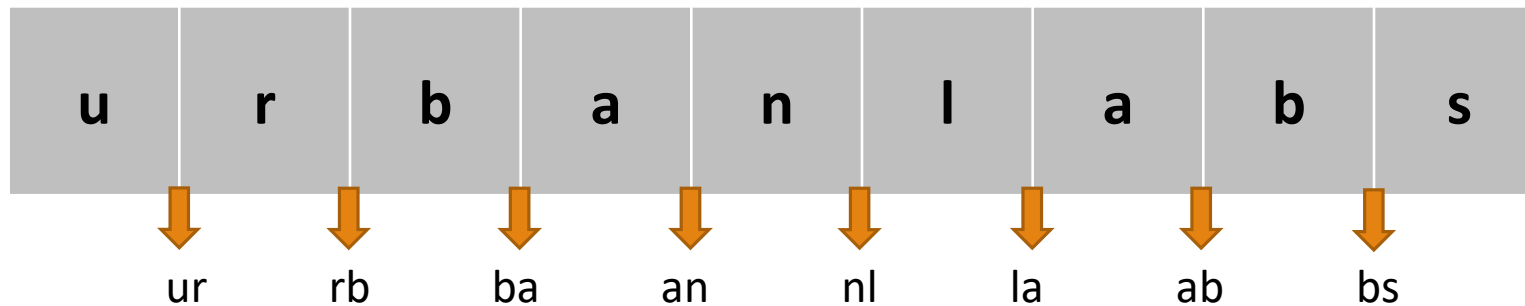
- *But wait, if the hash function produces the same value for a string every time, won't it be easy to reverse the algorithm or keep track of values that correspond to particular words?*
  - Cryptographic functions are designed to be **non-reversible**
  - One type of brute-force decryption involves “rainbow tables”
- This is where a ‘**salt**’ becomes useful (hidden/random value appended to the string before hashing)



# Protocol (from CCAC project)

---

- MD5, 'Salted', and Locality-Sensitive Hashing
- Locality-Sensitive Hashing (LSH)
  - **MinHash** (min-wise independent permutations)
  - Collect **minimum values** from hashing bigrams of the identifiable information (name), each time changing the hash value permutations by adding the iteration number to the
    - 150



# Protocol (from previous project)

---

- **Required Preparation**

- Date-shifting and Dataset Truncation

- Prevent identification/record linkage by date or ordering
    - **Shift all dates in the dataset by a random number of days** (to preserve temporal relationship)
    - Truncate random number of data points from front and back

- The Salt

- **Keyword, to be known only by analysts hashing the datasets**
    - Preferably read salt into script from another secure file

- Password-Protected Folders

- For data transfer between analysts
    - Use **randomly generated password**

# Protocol (from previous project)

---

- Analyst 1 handles sensitive dataset from Organization 1
  - Create a unique ID for each row
  - Truncate and Date-shift as necessary
  - Extract the name field and ID field into a separate dataset (Dataset A)
  - Extract the ID field and the other variables of interest into a separate dataset (Dataset B)
- Analyst 1 then communicates the salt and the value of the date-shift to Analyst 2 securely
- Analyst 2 handles sensitive Dataset from Organization
  - Perform the same steps as the dataset from Organization 1 to get Dataset C and Dataset D
- Analyst 1 runs the hashing script on Dataset A, drops the name column (Dataset E)
- Analyst 2 runs the hashing script on Dataset C, drops the name column (Dataset F)

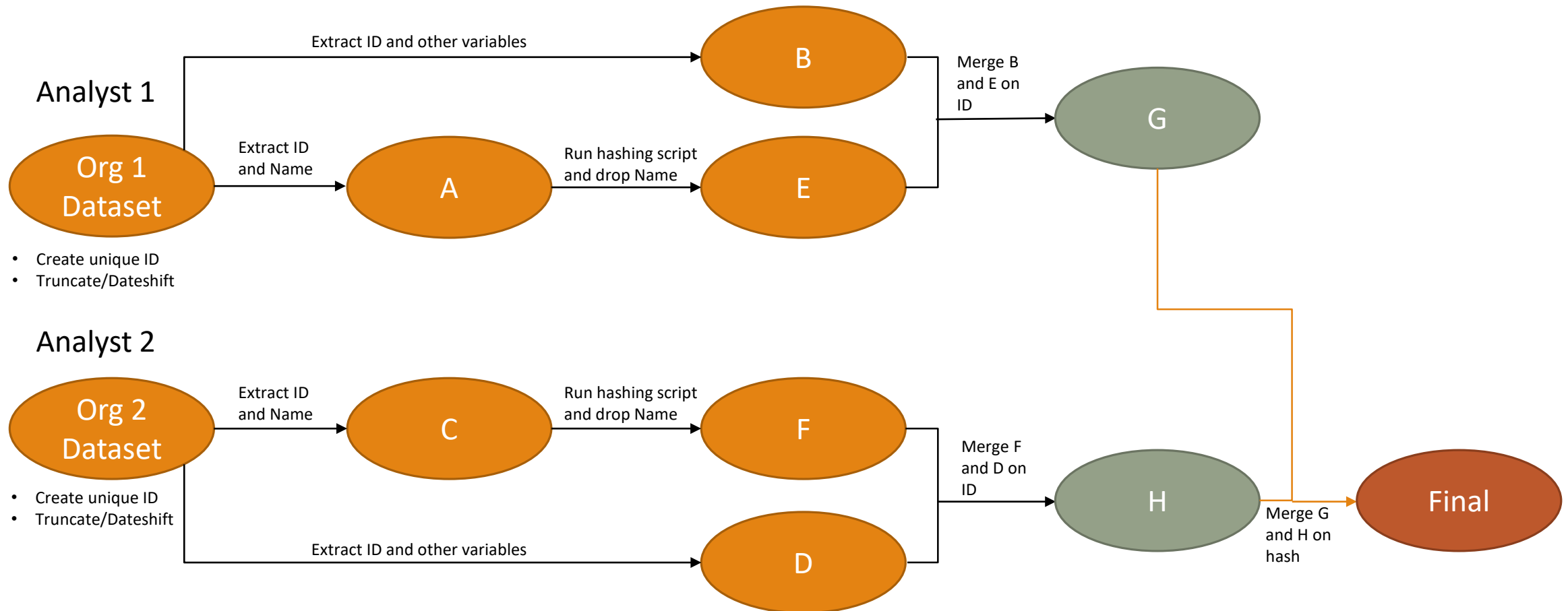


# Protocol (from previous project)

---

- Analyst 1 merges Dataset E with Dataset B using the unique ID (Dataset G)
- Analyst 2 merges Dataset F with Dataset D using the unique ID (Dataset H)
- Now, Dataset G is a dataset from Organization 1 with hashed names and Dataset H is a dataset from Organization 2 with hashed names
- Finally, Analyst 1 securely transfers Dataset G to Analyst 2, who can then perform the merging using the hashed names!

# Protocol (from previous project)



# Actual Code

---

# Code Flow

---

- run script with **python gen\_hash.py [csv filename] 150**
- code reads in the csv, stores name column and other variables into separate lists
  - cleans the names by lowercasing and removing all spaces
- code splits each name into **2-shingles**
- for each 2-shingle, the **salt is appended to the front and back, the iteration number, and then a hash value is obtained**
- in each iteration till 150, we take the **minimum of the these values**
- we end up with a series of **150 minhashes, which we call the signature.**
- code then outputs the names, cleaned names, the hash signature, and other relevant variables into a csv
- **remember to drop the names afterwards!** They are there just for checking purposes!

# Merging

---

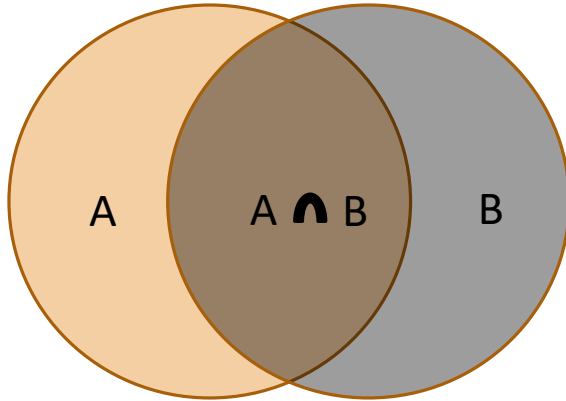
- Compare the hash signatures (vector of 150 hashes)
- As mentioned, two strings which are exactly the same should produce the exact same vector

Leona'rd Bloo#mfield	[4627305473336, 5086361082474, 19804513338981, 1
Wilhelm Wundt#	[4627305473336, 2940959422940, 5011493785980, 88
William James	[49637563509771, 3743988101286, 27797746129796,

# Merging

---

- Jaccard Similarity



$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Shared values / Total unique values

Currently, we have used a modified similarity score that is the **intersection divided by the width (150)**, which gives us a more intuitive score closer to 1. Basically this measures the ratio of shared values to non-shared ones.

# Merging

---

- Optimal threshold is difficult to determine (we've tried 0.9, 0.95)
- Currently in R, but we can also develop a Python script for it
  - And vice versa, the hashing is done in Python, but it may be possible to port it to other programming languages (There's a version made for SQL Studio, with the help of Dejan)
- Next week, we'll see if we can try it out on a sample dataset!

# Validation

---

- Difficult to definitively validate due to impossibility of accessing ground truth (viewing true names from both datasets)
- Proxy validation with “fake data”
  - Try to merge two hashed datasets of created names with some variation
  - Check if merging works as intended

ID	NAME	HASH (DEJAN)	HASH (LEOSON)	EQUAL
1	Leona'rd Bloo#mfield	[4627305473336, 5086361082474, 19804513338981, 1573]	[4627305473336, 5086361082474, 19804513338981, 1573]	TRUE
2	Wilhelm Wundt#	[4627305473336, 2940959422940, 5011493785980, 88145]	[4627305473336, 2940959422940, 5011493785980, 88145]	TRUE