|  |  |
| --- | --- |
| **Product Management User Stories** |  |
| **GitLab Repo Link** | <https://gitlab.wirecard-cloud.com/data_science/consumer_identification.git> |
| **Test Datasets** | Input: R:\Data Services and BI\Staff\03\_DataProducts\07\_Repository\Test\_Datasets\input\ConsID\_\*.csv  Output: R:\Data Services and BI\Staff\03\_DataProducts\07\_Repository\Test\_Datasets\output\ConsID\_\*.csv |
| [**README.md**](http://readme.md/) | <https://gitlab.wirecard-cloud.com/data_science/consumer_identification/blob/master/README.md> |

Offline Algorithm

Advantages of accurate consumer identification:

-          Provides the opportunity for detailed and informative analysis

-         Improves predictions of customer behavior.

**The input**

 The algorithm receives data of transactions. Relevant column required for the procedure are the following.

(Note: The following are the names of the global variables defined in the file globals.py; see GitLab Repo)

PAYMENT\_METHOD, CUSTOMER\_IBAN, CUSTOMER\_FIRST\_NAME, CUSTOMER\_LAST\_NAME, CUSTOMER\_CARD\_EXPIRY\_DATE,

CUSTOMER\_EMAIL, CUSTOMER\_PAN, CUSTOMER\_CARD\_HOLDER\_NAME, TRANSACTION ID

All of the above fields could be anonymized/hashed/tokenized/etc. This would not affect the algorithm nor the resulting output.

**The Output**

-          A data frame containing the keys for all customers linked to the customer’s ID.

-          A data frame containing all transactions ids and the customer ids associated with them

**Algorithm**

The workflow of the offline customer identification algorithm is described and the steps are illustrated with a simple example.

(Note: All information used in the example is dummy data.)

**1.      Construct a payment method key for each transaction in the dataset.**

 The transactions are filtered to remove all unrecognized payment methods. Logging message provides the information of the number of dropped transactions.

 The function **get\_unique\_customer\_id\_per\_payment\_method** creates a payment method key for each transaction.

The input to the function is a row the data frame of transaction. The output **key** is created based on the payment method and the customers’ details ([Reference](https://confluence.wirecard.sys/#ConsumerIdentification-Documentation-PaymentMethodKeys)).

Each combination of payment method and customer details receives a key. A column containing all the keys is appended to the original data frame of transactions

|  |  |  |  |
| --- | --- | --- | --- |
| **Transactional data** | **Email** | **Payment Method** | **PAYMENT\_METHOD\_KEY** |
|  | [johndoe@mail.com](mailto:johndoe@mail.com) | CreditCard1 | “0164321-johndoe-09/21’’ |
|  | [johndoe@mail.com](mailto:johndoe@mail.com) | CreditCard2 | “2335256-johndoe-01/23’’ |
|  | [alice01@mail.com](mailto:alice01@mail.com) | CreditCard2 | “2335256-johndoe-01/23’’ |
|  | [alice01@mail.com](mailto:alice01@mail.com) | CreditCard3 | “7135256-bobdoe-06/23’’ |
|  | [alice01@mail.com](mailto:alice01@mail.com) | Paypal | “alice01@[mail.co](http://mail.com/)m |
|  | [bob@mail.com](mailto:bob@mail.com) | CreditCard3 | “7135256-bobdoe-06/23’’ |
|  | [bob@mail.com](mailto:bob@mail.com) | SEPA Credit | “de5301000389238786” |

**2.      Assign a customer id and payment\_method\_id to each payment method key.**

For efficiency purposes, a temporary data frame which contains all unique payment method key is created, Using the indices of the rows in this data frame, each key is being associated with payment method id and customer id (initially equal).

The data frame is then joined with the datafremae of transactions on payment method KEY*.*As a result, all identical payment method keys have the same KEY\_ID and CUSTOMER\_ID assigned.

|  |  |  |  |
| --- | --- | --- | --- |
| **EMAIL** | **KEY** | **CUSTOMER\_ID** | **KEY\_ID** |
| [johdoe@mail.com](mailto:johdoe@mail.com) | “0164321-johndoe-09/21’’ | 1 | 1 |
| [johndoe@mail.com](mailto:johndoe@mail.com) | “2335256-johndoe-01/23’’ | 2 | 2 |
| [alice01@mail.com](mailto:alice01@mail.com) | “2335256-johndoe-01/23’’ | 2 | 2 |
| [alice01@mail.com](mailto:alice01@mail.com) | “7135256-bobdoe-06/23’’ | 3 | 3 |
| [alice01@mail.com](mailto:alice01@mail.com) | “alice01@[mail.com](http://mail.com/)” | 4 | 4 |
| [bob@mail.com](mailto:bob@mail.com) | “7135256-bobdoe-06/23’’ | 3 | 3 |
| [bob@mail.com](mailto:bob@mail.com) | “de5301000389238786” | 5 | 5 |

**3.      Identify the unique customers**

 The following two operations represent the main functionality of the algorithm. They are repeated in a loop which terminates when the number of identified customers stops decreasing.

**3.1**  First, the transactions are grouped by customer’s EMAIL where provided. Transactions with identical email address are assigned the same Customer\_ID which is the minimum of all of the IDs for the group.

Transactions with no match found or missing email remain unchanged.

**Iteration: 1**

|  |  |  |
| --- | --- | --- |
| All transactional data | EMAIL | CUSTOMER\_ID |
|  | [johndoe@mail.com](mailto:johndoe@mail.com) | 1 |
|  | [johndoe@mail.com](mailto:johndoe@mail.com) | ~~2~~ 1 |
|  | [alice01@mail.com](mailto:alice01@mail.com) | 2 |
|  | [alice01@mail.com](mailto:alice01@mail.com) | ~~3~~ 2 |
|  | [alice01@mail.c](mailto:alice01@mail.com)om | ~~4~~ 2 |
|  | [bob@mail.com](mailto:bob@mail.com) | 3 |
|  | [bob@mail.com](mailto:bob@mail.com) | ~~5~~ 3 |

**3.2**    Next, all transactions are grouped by the payment method key (KEY\_ID). Following the same principle, each group receive a common CUSTOMER\_ID which is the minimum of the group’s ids.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Transactional Data** | **EMAIL** | **KEY** | **KEY\_ID** | **CUSTOMER\_ID** |
|  | [johdoe@mail.com](mailto:johdoe@mail.com) | “0164321-johndoe-09/21’’ | 1 | 1 |
|  | [johndoe@mail.com](mailto:johndoe@mail.com) | “2335256-johndoe-01/23’’ | 2 | 1 |
|  | [alice01@mail.com](mailto:alice01@mail.com) | “2335256-johndoe-01/23’’ | 2 | ~~2~~ 1 |
|  | [alice01@mail.com](mailto:alice01@mail.com) | “7135256-bobdoe-06/23’’ | 3 | 2 |
|  | [alice01@mail.com](mailto:alice01@mail.com) | “alice01@[mail.com](http://mail.com/)” | 4 | 2 |
|  | [bob@mail.com](mailto:bob@mail.com) | “7135256-bobdoe-06/23’’ | 3 | ~~3~~ 2 |
|  | [bob@mail.com](mailto:bob@mail.com) | “de5301000389238786” | 5 | 3 |

The above two-step procedure (3.1, 3.2) is repeated until all transactions with the same email and payment method key are associated with a common CUSTOMER\_ID, i.e. until the number of identified customers stops decreasing.

Next Iterations:

**Iteration: 2**

 Group by email. Group by payment method id (KEY).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Transactional Data** | **EMAIL** | **KEY** | **KEY\_ID** | **CUSTOMER\_ID** |
|  | [johdoe@mail.com](mailto:johdoe@mail.com) | “0164321-johndoe-09/21’’ | 1 | 1 |
|  | [johndoe@mail.com](mailto:johndoe@mail.com) | “2335256-johndoe-01/23’’ | 2 | 1 |
|  | [alice01@mail.com](mailto:alice01@mail.com) | “2335256-johndoe-01/23’’ | 2 | 1 |
|  | [alice01@mail.com](mailto:alice01@mail.com) | “7135256-bobdoe-06/23’’ | 3 | ~~2~~ 1 |
|  | [alice01@mail.com](mailto:alice01@mail.com) | “alice01@[mail.com](http://mail.com/)” | 4 | ~~2~~ 1 |
|  | [bob@mail.com](mailto:bob@mail.com) | “7135256-bobdoe-06/23’’ | 3 | ~~2~~ 1 |
|  | [bob@mail.com](mailto:bob@mail.com) | “de5301000389238786” | 5 | ~~3~~ 2 |

**Iteration: 3**

 Group by email. Group by payment method id –> no change.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Transactional Data** | **EMAIL** | **KEY** | **KEY\_ID** | **CUSTOMER\_ID** |
|  | [johdoe@mail.com](mailto:johdoe@mail.com) | “0164321-johndoe-09/21’’ | 1 | 1 |
|  | [johndoe@mail.com](mailto:johndoe@mail.com) | “2335256-johndoe-01/23’’ | 2 | 1 |
|  | [alice01@mail.com](mailto:alice01@mail.com) | “2335256-johndoe-01/23’’ | 2 | 1 |
|  | [alice01@mail.com](mailto:alice01@mail.com) | “7135256-bobdoe-06/23’’ | 3 | 1 |
|  | [alice01@mail.com](mailto:alice01@mail.com) | “alice01@[mail.com](http://mail.com/)” | 4 | 1 |
|  | [bob@mail.com](mailto:bob@mail.com) | “7135256-bobdoe-06/23’’ | 3 | 1 |
|  | [bob@mail.com](mailto:bob@mail.com) | “de5301000389238786” | 5 | ~~2~~ 1 |

**4.      Return the resulting data frames**

The function returns two data frames:

* One contains all CUSTOMER\_ID and corresponding KEY which is either email or payment method KEY.
* One containing all TRANSACTION\_IDs and the CUSTOMER\_ID linked to them.

|  |  |
| --- | --- |
| **CUSTOMER\_ID** | **KEY** |
| 1 | “0164321-johndoe-09/21’’ |
| 1 | “2335256-johndoe-01/23’’ |
| 1 | “alice01@[mail.com](http://mail.com/)” |
| 1 | “de5301000389238786” |
| 1 | [johndoe@mail.com](mailto:johndoe@mail.com) |
| 1 | [alice01@mail.com](mailto:alice01@mail.com) |
| 1 | bob[@mail.com](mailto:alice01@mail.com) |

|  |  |
| --- | --- |
| **CUSTOMER\_ID** | **TRANSACTION\_ID** |
| 1 | 1 |
| 1 | 2 |
| 1 | 3 |
| 1 | 4 |
| 1 | 5 |
| 1 | 6 |

Online Algorithm

**Overview**

This section explains in its totality the functionality of the online customer identification algorithm. We will now proceed to describe its steps to be taken if we are to achieve the same result from the offline customer identification.

Note that this algorithm is intended to process only one transaction at a given time. A batch of transactions can also be given as an input, the performance however will be significantly poorer than the one from the offline customer identification.

**Input**

The input to this algorithm will be two data sets. The first data set is the set of transactions from where the customers will be obtained. The following columns are necessary in this data set for the algorithm to execute correctly:

PAYMENT\_METHOD, CUSTOMER\_PAN, CUSTOMER\_CARD\_HOLDER\_NAME, CUSTOMER\_CARD\_EXPIRY\_DATE, CUSTOMER\_IBAN, CUSTOMER\_EMAIL, CUSTOMER\_FIRST\_NAME, CUSTOMER\_LAST\_NAME.

A special remark is made on the PAYMENT\_METHOD column. Just as with the offline customer identification algorithm we process currently only 11 payment methods. If a transaction with a different payment method is identified, this is dropped out from the transactions dataframe before the algorithm starts.

The second data set is the current database of customers with columns CUSTOMER\_ID and KEY. The latter can be empty or the output from either the offline customer identification algorithm or a previous run of the online customer identification algorithm.

**Output**

The output of the algorithm will be the updated customer database and the transactions data set with a new column named CUSTOMER ID, which will contain the id, also found in the customer database, of the customer who executed the transaction.

**Steps**

1. **For the new transaction, create a new column called Keys. This column will contain a list with all the possible keys that can help identify a unique customer of a transaction.This keys are the CUSTOMER\_EMAIL if available and the key corresponding to the PAYMENT\_METHOD of the transaction. See the section Payment Method Keys for more details.**
2. **Once the keys are assigned we search in our customer database for all the customer IDs that have been already associated with one of them.**
3. **If none Customer IDs are found, we identified a new customer and proceed to step 4, else we have previously identified this customer and we proceed to step 7.**
4. **We look in our database the current maximum id and we use its successor as the ID for our new customer.**
5. **For each key of the transaction add a new row in the Customer database using as Customer ID the ID obtained at step 4.**
6. **Go to step 12.**
7. **If we have only one customer ID means that there is only one customer and we proceed to step 8, otherwise it means that previously we thought we had at least 2 different customers but it turns out there are the same and we proceed to step 10.**
8. **For each key in the transaction that is not yet in the database we add a new row into the database using the Customer ID obtained in step 2 as the Customer ID.**
9. **Go to step 12.**
10. **From the list of customers IDs obtained at 2 we take the minimum one and in the database we replace the all of them with this value.**
11. **For each key in the transaction that is not currently in the database we add a new row into the database using the Customer ID obtained in step 10 as the Customer ID.**
12. **End of the algorithm.**

**Example**

We now make an example to make the algorithm clearer. We will take the same example from the offline customer identification. We receive a series of transactions made by one customer with the following payment methods. We will assume that our customer database is empty at the beginning of the algorithm.

|  |  |  |
| --- | --- | --- |
| **Transaction** | **EMAIL** | **Payment Method** |
| TX 1 | [johdoe@mail.com](mailto:johdoe@mail.com) | CreditCard1 |
| TX 2 | [johdoe@mail.com](mailto:johdoe@mail.com) | CreditCard2 |
| TX 3 | [alice01@mail.com](mailto:alice01@mail.com) | CreditCard2 |
| TX 4 | [alice01@mail.com](mailto:alice01@mail.com) | CreditCard3 |
| TX 5 | [alice01@mail.com](mailto:alice01@mail.com) | Paypal |
| TX 6 | [bob@mail.com](mailto:bob@mail.com) | CreditCard3 |
| TX 7 | [bob@mail.com](mailto:bob@mail.com) | SEPA Credit |

We start with TX 1. First we assign its possible keys in the new column Keys

|  |  |  |  |
| --- | --- | --- | --- |
| **Transaction** | **EMAIL** | **Payment Method** | **Keys** |
| TX 1 | [johdoe@mail.com](mailto:johdoe@mail.com) | CreditCard1 | [[johdoe@mail.com](mailto:johdoe@mail.com), "0164321-johndoe-09/21"] |

We look in the customer database for Customer IDs, which are already assigned to the following keys. Since our database is empty, there will be no keys and we proceed to step 4.

**Customer IDs = Empty**

We look for the successor of the maximum ID of the database. Since the database is empty we select the ID as 1. We go to step 5.

**New Customer ID = 1**

We use these new Customer ID to insert in the database all of the keys of this transaction. The database will look like:

|  |  |
| --- | --- |
| **Customer ID** | **Key** |
| 1 | [johdoe@mail.com](mailto:johdoe@mail.com) |
| 1 | "0164321-johndoe-09/21" |

We now proceed to process TX 2. We assign to the transaction its Keys.

|  |  |  |  |
| --- | --- | --- | --- |
| **Transaction** | **EMAIL** | **Payment Method** | **Keys** |
| TX 2 | [johdoe@mail.com](mailto:johdoe@mail.com) | CreditCard2 | [[johdoe@mail.com](mailto:johdoe@mail.com), “2335256-johndoe-01/23’’] |

We look in the customer database for Customer IDs, which are already assigned to the keys. Since [johdoe@mail.com](mailto:johdoe@mail.com) is already in the database we will obtain the Customer ID 1 and we proceed to step 7.

**Customer IDs = 1**

We see that we only have one Customer ID and we proceed to step 8. We insert the key “2335256-johndoe-01/23’ in the database with the Customer ID 1. The database will look like:

|  |  |
| --- | --- |
| **Customer ID** | **Key** |
| 1 | [johdoe@mail.com](mailto:johdoe@mail.com) |
| 1 | "0164321-johndoe-09/21" |
| 1 | “2335256-johndoe-01/23’’ |

We now proceed to TX 3. We assign to the transaction its Keys

|  |  |  |  |
| --- | --- | --- | --- |
| **Transaction** | **EMAIL** | **Payment Method** | **Keys** |
| TX 3 | [alice01@mail.com](mailto:alice01@mail.com) | CreditCard2 | [[alice01@mail.com](mailto:alice01@mail.com), “2335256-johndoe-01/23’’] |

We look in the customer database for Customer IDs, which are already assigned to the keys. “2335256-johndoe-01/23’ is already in the database we will obtain the Customer ID 1 and we proceed to step 7.

**Customer IDs = 1**

We see that we only have one Customer ID and we proceed to step 8. We insert the key [alice01@mail.com](mailto:alice01@mail.com) in the database with the Customer ID 1. The database will look like:

|  |  |
| --- | --- |
| **Customer ID** | **Key** |
| 1 | [johdoe@mail.com](mailto:johdoe@mail.com) |
| 1 | "0164321-johndoe-09/21" |
| 1 | “2335256-johndoe-01/23’’ |
| 1 | [alice01@mail.com](mailto:alice01@mail.com) |

We now proceed to TX 4. We assign to the transaction its Keys

|  |  |  |  |
| --- | --- | --- | --- |
| **Transaction** | **EMAIL** | **Payment Method** | **Keys** |
| TX 4 | [alice01@mail.com](mailto:alice01@mail.com) | [alice01@mail.com](mailto:alice01@mail.com) | [Email 2, “7135256-bobdoe-06/23’’] |

We look in the customer database for Customer IDs, which are already assigned to the keys. [alice01@mail.com](mailto:alice01@mail.com) is already in the database we will obtain the Customer ID 1 and we proceed to step 7.

**Customer IDs = 1**

We see that we only have one Customer ID and we proceed to step 8. We insert the key “7135256-bobdoe-06/23’’ in the database with the Customer ID 1. The database will look like:

|  |  |
| --- | --- |
| **Customer ID** | **Key** |
| 1 | [johdoe@mail.com](mailto:johdoe@mail.com) |
| 1 | "0164321-johndoe-09/21" |
| 1 | “2335256-johndoe-01/23’’ |
| 1 | [alice01@mail.com](mailto:alice01@mail.com) |
| 1 | “7135256-bobdoe-06/23’’ |

We now proceed to TX 5. We assign to the transaction its Keys

|  |  |  |  |
| --- | --- | --- | --- |
| **Transaction** | **EMAIL** | **Payment Method** | **Keys** |
| TX 5 | [alice01@mail.com](mailto:alice01@mail.com) | Paypal | [[alice01@mail.com](mailto:alice01@mail.com)] |

We look in the customer database for Customer IDs, which are already assigned to the keys. [alice01@mail.com](mailto:alice01@mail.com) is already in the database we will obtain the Customer ID 1 and we proceed to step 7.

**Customer IDs = 1**

We see that we only have one Customer ID and we proceed to step 8. Since all keys are already in the database we don’t add new rows. The database will look like:

|  |  |
| --- | --- |
| **Customer ID** | **Key** |
| 1 | [johdoe@mail.com](mailto:johdoe@mail.com) |
| 1 | "0164321-johndoe-09/21" |
| 1 | “2335256-johndoe-01/23’’ |
| 1 | [alice01@mail.com](mailto:alice01@mail.com) |
| 1 | “7135256-bobdoe-06/23’’ |

We now proceed to TX 6.  We assign to the transaction its Keys

|  |  |  |  |
| --- | --- | --- | --- |
| **Transaction** | **EMAIL** | **Payment Method** | **Keys** |
| TX 6 | [bob@mail.com](mailto:bob@mail.com) | CreditCard3 | [[bob@mail.com](mailto:bob@mail.com), “7135256-bobdoe-06/23’’] |

We look in the customer database for Customer IDs, which are already assigned to the keys.  “7135256-bobdoe-06/23’’ is already in the database we will obtain the Customer ID 1 and we proceed to step 7.

**Customer IDs = 1**

We see that we only have one Customer ID and we proceed to step 8. We insert the key [bob@mail.com](mailto:bob@mail.com) in the database with the Customer ID 1. The database will look like:

|  |  |
| --- | --- |
| **Customer ID** | **Key** |
| 1 | [johdoe@mail.com](mailto:johdoe@mail.com) |
| 1 | "0164321-johndoe-09/21" |
| 1 | “2335256-johndoe-01/23’’ |
| 1 | [alice01@mail.com](mailto:alice01@mail.com) |
| 1 | “7135256-bobdoe-06/23’’ |
| 1 | [bob@mail.com](mailto:bob@mail.com) |

We now proceed to TX 7. We assign to the transaction its Keys

|  |  |  |  |
| --- | --- | --- | --- |
| **Transaction** | **EMAIL** | **Payment Method** | **Keys** |
| TX 7 | [bob@mail.com](mailto:bob@mail.com) | SEPA Credit | [[bob@mail.com](mailto:bob@mail.com)] |

We look in the customer database for Customer IDs, which are already assigned to the keys. [bob@mail.com](mailto:bob@mail.com) is already in the database we will obtain the Customer ID 1 and we proceed to step 7.

**Customer IDs = 1**

We see that we only have one Customer ID and we proceed to step 8. Since all keys are already in the database we don’t add new rows. The database will look like:

|  |  |
| --- | --- |
| **Customer ID** | **Key** |
| 1 | [johdoe@mail.com](mailto:johdoe@mail.com) |
| 1 | "0164321-johndoe-09/21" |
| 1 | “2335256-johndoe-01/23’’ |
| 1 | [alice01@mail.com](mailto:alice01@mail.com) |
| 1 | “7135256-bobdoe-06/23’’ |
| 1 | [bob@mail.com](mailto:bob@mail.com) |

And our customer database is the same as the one obtained with the offline customer identification.

**Order Invariability**

Now let’s assume that the order of TX 5 and TX 7 is exchanged

|  |  |  |
| --- | --- | --- |
| **Transaction** | **EMAIL** | **Payment Method** |
| TX 1 | [johdoe@mail.com](mailto:johdoe@mail.com) | CreditCard1 |
| TX 2 | [johdoe@mail.com](mailto:johdoe@mail.com) | CreditCard2 |
| TX 3 | [alice01@mail.com](mailto:alice01@mail.com) | CreditCard2 |
| TX 4 | [alice01@mail.com](mailto:alice01@mail.com) | CreditCard3 |
| TX 7 | [bob@mail.com](mailto:bob@mail.com) | SEPA Credit |
| TX 6 | [bob@mail.com](mailto:bob@mail.com) | CreditCard3 |
| TX 5 | [alice01@mail.com](mailto:alice01@mail.com) | Paypal |

Until TX 4, the algorithm’s run is the same and our customer database will look like:

|  |  |
| --- | --- |
| **Customer ID** | **Key** |
| 1 | [johdoe@mail.com](mailto:johdoe@mail.com) |
| 1 | "0164321-johndoe-09/21" |
| 1 | “2335256-johndoe-01/23’’ |
| 1 | [alice01@mail.com](mailto:alice01@mail.com) |
| 1 | “7135256-bobdoe-06/23’’ |

We now proceed to TX 7. We assign to the transaction its Keys

|  |  |  |  |
| --- | --- | --- | --- |
| **Transaction** | **EMAIL** | **Payment Method** | **Keys** |
| TX 7 | [bob@mail.com](mailto:bob@mail.com) | SEPA Credit | [[bob@mail.com](mailto:bob@mail.com)] |

We look in the customer database for Customer IDs, which are already assigned to the keys. [bob@mail.com](mailto:bob@mail.com) is not yet in the database so no Customer ID will be recovered. We proceed then to step 4

**Customer IDs = None**

We look for the successor of the maximum ID of the database. In this case this ID will be the number 2. We go to step 5.

**New Customer ID = 2**

We use these new Customer ID to insert in the database all of the keys of this transaction. The database will look like:

|  |  |
| --- | --- |
| **Customer ID** | **Key** |
| 1 | [johdoe@mail.com](mailto:johdoe@mail.com) |
| 1 | "0164321-johndoe-09/21" |
| 1 | “2335256-johndoe-01/23’’ |
| 1 | [alice01@mail.com](mailto:alice01@mail.com) |
| 1 | “7135256-bobdoe-06/23’’ |
| 2 | [bob@mail.com](mailto:bob@mail.com) |

 We now proceed to TX 6.  We assign to the transaction its Keys

|  |  |  |  |
| --- | --- | --- | --- |
| **Transaction** | **EMAIL** | **Payment Method** | **Keys** |
| TX 6 | [bob@mail.com](mailto:bob@mail.com) | CreditCard3 | [[bob@mail.com](mailto:bob@mail.com), “7135256-bobdoe-06/23’’] |

We look in the customer database for Customer IDs, which are already assigned to the keys.  “7135256-bobdoe-06/23’’ and [bob@mail.com](mailto:bob@mail.com) are already in the database we will obtain the Customer IDs 1 and 2 and we proceed to step 7.

**Customer IDs = [1, 2]**

We see that we only have two Customer IDs. This means that customers 1 and 2 are indeed the same customer and we proceed to step 10. We obtain the minimum of the customer IDs obtained and replace the other with this value in the database.

**Minimum Customer ID = 1**

The database will now look like

|  |  |
| --- | --- |
| **Customer ID** | **Key** |
| 1 | [johdoe@mail.com](mailto:johdoe@mail.com) |
| 1 | "0164321-johndoe-09/21" |
| 1 | “2335256-johndoe-01/23’’ |
| 1 | [alice01@mail.com](mailto:alice01@mail.com) |
| 1 | “7135256-bobdoe-06/23’’ |
| 1 | [bob@mail.com](mailto:bob@mail.com) |

We now proceed to TX 5. We assign to the transaction its Keys

|  |  |  |  |
| --- | --- | --- | --- |
| **Transaction** | **EMAIL** | **Payment Method** | **Keys** |
| TX 5 | [alice01@mail.com](mailto:alice01@mail.com) | Paypal | [[alice01@mail.com](mailto:alice01@mail.com)] |

We look in the customer database for Customer IDs, which are already assigned to the keys. [alice01@mail.com](mailto:alice01@mail.com) is already in the database we will obtain the Customer ID 1 and we proceed to step 7.

**Customer IDs = 1**

We see that we only have one Customer ID and we proceed to step 8. Since all keys are already in the database we don’t add new rows. The database will look like:

|  |  |
| --- | --- |
| **Customer ID** | **Key** |
| 1 | [johdoe@mail.com](mailto:johdoe@mail.com) |
| 1 | "0164321-johndoe-09/21" |
| 1 | “2335256-johndoe-01/23’’ |
| 1 | [alice01@mail.com](mailto:alice01@mail.com) |
| 1 | “7135256-bobdoe-06/23’’ |
| 1 | [bob@mail.com](mailto:bob@mail.com) |

And we obtain the same result. This is a small example showing that the order in which the transactions are processed does not change the final result.

Payment Method Keys

The following table summarizes the way each payment method KEY is created.

 Fields containing "Nothing" denote the payment methods which does not provide enough information for the customer, hence no payment method key could be created.

Transactions with these payment methods, will be filtered prior to the consumer identification procedure(See [Algorithm](https://confluence.wirecard.sys/#ConsumerIdentification-Documentation-Algorithm) description).

|  |  |
| --- | --- |
| **Payment Method** | **Key Information** |
| Guaranteed Invoice | email |
| Paypal | email |
| Przelwy24 | email |
| iDEAL | IBAN |
| CARD | Pan + Card Holder Name + Card Expiry Date |
| Credit Card | Pan + Card Holder Name + Card Expiry Date |
| Soforüberweisung | IBAN |
| Sepa CreditBancontact | IBAN |
| EPS | IBAN |
| SEPA Direct Debit | IBAN |
| Bancontact | IBAN + Customer First Name+ Customer Last Name |
| paysafecard | Nothing |
| Alipay Spot Pay | Wallet Account ID |
| voucher | Nothing |
| Wechat Quick Pay | Nothing |
| Maybank2u | Nothing |
| CIMB Click | Nothing |
| Giropay | IBAN |
| Alipay Cross-Border E-Payment Service | email |
| Paydirekt | Nothing |
| MasterPass | email |
| Barzahlen | Nothing |
| Trustly | email |
| POLi | IBAN+BIC (IBAN actually contains account number, IBAN can be determined using account number and BIC.) |
| Zapp | Nothing |
| Bitcoin | Wallet Account ID |
| Klarna Guaranteed Invoice | email |
| UnionPay Online Payment | email |
| Klarna Guaranteed Installments | email |
| Guaranteed Direct Debit | IBAN |
| TrustPay | email |
| Payolution Invoice | email |
| Skrill | email |
| Wechat QRPay | email |
| Payolution B2B | email |
| [Moneta.ru](http://moneta.ru/) | email |