Department of Computer Science

Submitted in part fulfilment for the degree of BEng.

Call Data Record Correlation

Thomas Thorpe

2019-April-30

Supervisor: Angus M. Marshall

To the most loved member of the family; Bottles the cat

Acknowledgements

Thanks to Angus Marshall for being a fantastic supervisor.
Thanks to all my family for the continuous support throughout my degree.

Contents

List of Figures

List of Tables

Executive Summary

The telephone network is used by millions to communicate with others daily whenever and wherever due to the rise of mobile phones and the wide coverage of mobile providers. Although the rise in connectivity has improved the lives of many and drastically increased productivity in business due to instant communication, organised crime has also taken advantage of these benefits.

As a result law enforcement often need to prove that communication has occurred between people to find the associates of someone in the gang. Fortunately, a record of all calls that take place on the telephone network are recorded by the Mobile Network Operators which can be accessed by law enforcement if required. The data records about the calls vary from provider to provider but common fields include: starting and ending times, if the call was initiated or received along with the called and calling number.

Previously law enforcement could look at these records from the telephone numbers of the two people under investigation and confirm that calls had taken place between them. This would give law enforcement further reason to continue the investigation. However it was found that this simple analysis no longer produced reliable results and often pointed to innocent third party numbers or numbers which had not been allocated to anyone. Recently it has been discovered that gangs are using modified stealth SIMs to obscure the number fields that get recorded by Mobile Network Operators. Fortunately, other fields in the records cannot currently be spoofed or removed; namely the starting and ending times of the calls.

A timing-based heuristic has been developed based on a hypothesis of the infrastructure used by the gangs. This can identify when two records are the same call, regardless of the network mechanism used to obscure the numbers. However, this process requires a lot of labour to carry out manually and quickly becomes unwieldy as the number of records increases. Therefore, the aim of this project is to produce an automated tool which can identify calls between two people who are using these stealth SIMs, using this timing-based heuristic and other meta-data that cannot be spoofed. These results are then analysed alongside any other meaningful statistics which could be useful to law enforcement.

For this to be achieved the records have been standardised into a consistent format, as the data stored in the records varies between network

operators. The documentation is clear and concise so that it is possible for additional network providers to have a parser developed.

Once the input had been standardised, the tool was developed to automate the timing-based heuristic to identify records which are correlated. Furthermore, once these pairs of records were found, statistics such as the proportion of correlated calls and which mechanisms have been used are computed and presented to the user.

In order to ensure the correctness of the tool, both the standardisation of records and the timing analysis needed to be verified. First the standardisation was checked by systematically providing all expected formats of records to confirm the output was as expected. To verify the correctness of the timing analysis, an experimental methodology was adopted; synthetic data was generated so that the correlated pairs were known. Tests have then confirmed the correct pairs are identified.

The tool has been tested with inputs that contained either none, one or all correlated calls at a small scale to demonstrate the tool can detect the spoofing mechanisms used by the criminal gangs. Furthermore, the tool was tested on larger input sizes of up to ten thousand varying different parameters such as the gaps between calls and the proportion of correlated calls. The results showed that the tool correctly identified all the correlated pairs through the use of timing analysis. The implications of these results is that over period of time it is possible to identify calls between two people regardless of if stealth SIMs being used.

There are improvements that can build upon these results, such as carrying out network analysis between multiple members of the organised crime gangs to determine which members are contacted the most and how information flows through the gang. Other heuristics can be integrated into the tool by looking at other patterns such as how often the spoofed numbers are changed. Finally the tool could be further optimised to make larger inputs more feasible.

There are ethical considerations which concern this project due to the exposing nature of the tool. This is because there are legitimate use cases for spoofing numbers such as business numbers with an internal telephone system. Furthermore, there could be ethical considerations on the data used to verify the correctness of the tool due to using real numbers. However, as the numbers were generated randomly and only checked for matches between records by the tool, this is not a significant concern. Furthermore the data generation program could be adapted to map the numbers into the unallocated range.

1 Introduction

1.1 Background and Motivation

Telephone communication has played a significant part in the history of human communication and is still being widely used today with voice calls nearing ten billion a month [1]. As it is to be expected with any communication system that increases in size, criminal gangs have adopted this technology. The fact that the mobile network provides vast signal coverage over major cities means gangs can use mobiles to contact each other with ease.

The mobile network is made up of thousands of cell towers [2] that handsets connect to and start initiating a voice call. When this happens the handset shares identifiers known as the IMEI (International Mobile Equipment Identity) and IMSI (International Mobile Subscriber Identity). It is common practice for network providers to keep a log of all calls that go through their cell towers thereby allowing them to build up a collection of call records for a subscriber when they access the network.

For law enforcement it is possible to use these records to map out the connections between gang members by checking the CDR (Call Data Records) of each person they suspect and observing how often they phone one another. When combined with other investigation material such as approximate location information, deciding who is involved becomes a simpler problem.

However, recently law enforcement has found that these methods no longer work reliably as spoofed data has been found in the CDR logs. Sometimes this has directed investigators to innocent third parties, other times unissued numbers have been found which cannot be linked to a subscriber.

The working theory as to how spoofed numbers are being placed into the records is that stealth SIMs which contain extra programming are using the network management features; Call Redirection and Roaming Callback. Both of these have perfectly legitimate uses for companies and customers but can be abused to leave spoofed numbers in the CDRs. Moreover, it is not possible for law enforcement to obtain the records of the home network (HPLMN) as the infrastructure being used is in a country where it is difficult

to obtain data.

Assuming that this is the correct model, which seems likely given CDRs that have been analysed manually in the field, it is possible to uncover stealth calls by looking at the starting and ending times of the calls. This is due to the development of a timing-based heuristic developed by Angus Marshall and Peter Miller with law enforcement [3].

1.2 Aims and Objectives

I have expanded on the previous work by developing an automated tool to identify correlated calls when presented with the CDRs of suspects. Additionally the tool calculate metrics such as how many calls were found and which mechanisms have been used.

To achieve this the project needed to correctly parse the four main UK Mobile Network Operators (MNOs) into a standardised format. A further objective is to provide concise documentation on the format so that future parsers could be developed for other providers.

Another requirement is to create a tool flexible enough so that it can be adapted if a gang's infrastructure is changed in the future. For example the difference between starting times when the roaming callback mechanism might be randomised to be within a different time frame. An extended objective was to allow advanced analysis to be carried out such as network analysis if multiple members of the gang can be analysed.

1.3 Report Structure

The structure of the report is as follows:

Literature Review This chapter will provide the background knowledge required to carry out this project.

Design and Methodology This chapter will elicitate requirements to create a design for the tool and outline how the tool was tested. Furthermore, the implementation choices are compared and reviewed.

Results and Evaluation This chapter looks at the test results of the tool and evaluate the successfulness of the tool.

Conclusion This chapter will analyse the implications of the results and any further work that could be carried out.

1.4 Statement of Ethics

Lastly, there may be some ethical considerations in terms of the ability of the tool. There are legitimate reasons to use spoofing such as business usage for internal numbers. This tool has the ability to unmask these legitimate uses which is ethically bad. However for this tool to be used the CDRs of both participants are required which are kept by the MNOs. Furthermore, care needs to be taken when considering how the correctness of the tool was verified as the data required is phone numbers which could be allocated. In order to mitigate this, synthetic data can generated to test the tool without requiring the CDRs of real people. Finally the tool presents and saves all the results and metrics calculated for clerical review.

2 Literature Review

The purpose of this chapter is to provide a comprehensive view of any literature that is related to, or will be required for this project. This review will introduce key concepts and terminology.

Firstly, there is a section that provides a summary on the three components that are being utilized by the handsets of the gang members to provide a background on which fields of the call records can be spoofed and which are to be trusted.

From here, the review will look into semantic analysis and parsing of the CDRs from the four main UK mobile network providers. The tool will be required to preprocess the data by picking out the fields that are required for correlation and standardise the fields into a consistent format ready for record linkage using timing analysis.

Continuing on the review will look into how it is possible to pick out call records from two CDRs that could have spoofed numbers. This will be building on the methods designed in previous research [3] using rule-based record linkage using timing analysis.

Once these pairs of records can be identified, further heuristics and measurements will be brought forward which can find further patterns that the gang's infrastructure leaves behind. Further increasing the belief that communication is happening and being obscured.

Finally once this is the done, the review will look into different methods of analysing the results and how to display this to the user in a clear and concise way.

2.1 Stealth Calls Using USSD and CAMEL

There are two methods that can facilitate the use of stealth features such as spoofing numbers: Unstructured Supplementary Service Data (USSD) and Customised Applications for Mobile Enhanced Logic (CAMEL). These are both network mechanism features that allow a handset to communicate to their home network (HLPMN) to initiate call redirection or roaming callback which can result in spoofed numbers being placed in the CDR.

To enable these features, stealth SIMs contain CAMEL functions pre-programmed into the SIM Card which are initiated for any calls made or received by the phone. These SIMs are often accommodated by feature phones which obscure other identifying features of the device such as the IMEI and IMSI [4], forcing the highest encryption standard A5 [5], and preventing geolocation data being stored in Cell Towers.

2.1.1 Unstructured Supplementary Service Data

USSD is a network mechanism that allows the user and their home network (HPLMN) to communicate arbitrary data [6, page 7] which can be used for command and control. To send a USSD request back to the user's HPLMN the user has to dial a special code which activates services on the home network. These dial codes usually start with an asterisk (*) and end with a hash (#). Some common uses of USSD are moblie-money management services, prepaid callback service, menu based information services such as displaying whether forecasts.

The features that can be provided by USSD are provider-specific [6, Sec. 6.2.5] and hence it is possible that "USSD can be used to provide independent calling services such as a callback service" [7]. Furthermore these USSD codes could be used to change stealth features on the home network such as changing numbers required to dial for other spoofers or the number that appears in the Calling Line Identification (CLID) to the recipient.

USSD can be initiated from either the Mobile Station (MS) or the PLNM. There are many different entities in the mobile network such as the MSC, VLR, HLR however the USSD protocol is "transparent to the MS and to intermediate network entities" [7] as the USSD Handlers in the HLR and VLR "shall not store any information specific to the use of USSD" [6, Sec. 5.3 - 5.4]. Hence there is no metadata about whether USSD was used.

2.1.2 Customised Applications for Mobile Enhanced Logic

CAMEL is based on the Intelligent Network (IN) standards and is able to perform various actions when triggers happen, allowing logic into the network. CAMEL gives the home network the ability to both control call routing [8, Sec. 6] and SMS [8, Sec. 7].

CAMEL is a network feature designed "to help the network operator to provide the subscriber with the operator specific services even when roaming" [9]. This is useful for the infrastructure that supports stealth

SIMs as the SIM is always roaming with the HPLMN being out of law enforcements control. Furthermore CAMEL data is rarely captured by the base station the SIM is connected to; if the data is captured then this gives additional reason to believe spoofing is being applied to the call. There is the ability to handle call distribution via CAMEL which can be used to route calls over cheaper channels, or in the case at hand, over more private infrastructure [10] by requestion call redirection.

For these features to be available to the user the HPLMN has to have switchboard which can be used to route calls and place spoofed numbers in the CDR. This is not a limitation as it is possible to rent out virtual switch boards [11] and use Voice over IP (VoIP) to route the calls over the internet, an example is the SIP protocol [12].

2.1.3 Feature Phones and Stealth SIMs

Feature Phones are phones which through the use of additionally installed software can change the normally unchangeable IMEI of the phone [4], moreover there are many other features such as forcing higher encryption standard [13] and encrypting SMS [5]. Feature Phones are often best used in conjunction with Stealth SIMs described below.

Stealth SIMs are normal SIM cards that have modified to use a Virtual Mobile Network Operator (VMNO) which is controlled by the gang's infrastructure and acts as the HPLMN. This allows the SIMs to always to roaming when connecting to the telephone network. When aided by USSD and CAMEL the SIM can prevent the roaming network they are on from routing the calls and allow their HPLMN to handle the call. This information is never recorded by the roaming network and hence it is possible for their HPLMN to setup the call while the roaming network only sees the spoofed numbers [14]. As expected, these SIMs can hide other identifiable information such as the IMSI and can refuse to have encryption standard lowered below A5/1.

It is possible to buy both Feature Phones and Stealth SIMs online which promise to guarantee privacy with all the spoofing features mentioned above. The price of this equipment is normally in the range of £500+ which shows how valuable these modifications can be [14] [4].

2.2 Parsing and Semantic Analysis

Firstly to be able to begin record linkage to correlate calls, the data needs to be pre-processed and standardised. The objective of parsing and semantic

analysis is to "ensure that the attributes used for the matching have the same structure, and their content follows the same formats." [15, p24-25].

This requires understanding the semantic meaning of the headers for each provider to be able to identify inconsistencies in the way information about a CDR is encoded. Any misunderstanding here could result in errors when correlation is carried out however it is straight forward to make sure the correct elements are selected. Most providers have additional information that is not necessary to keep in the standardised format as they do not provide any information useful for correlation. To construct a standardised format one first has to decided upon the data fields that will be required to represent a call record that is impartial to network providers [16, Sec. 5.1].

The minimum information that is required to allow for correlation is:

Table 2.1: Standardised CDR

ID	Start Date & Time	End Date & Time	Type of Call	Calling Number	Called Number

Sometimes there can be additional fields depending on the network provider that can give further indication if calls are correlated such as if any CAMEL data was relayed. Furthermore some MNOs log other information such as the cell location at the stand and ends on the call. Although more complex, these other fields could also be used to identify other patterns which are indicative of spoofing and provide other useful information to law enforcement.

2.3 Correlation Heuristics

First, timing analysis is carried out to identify the pairs of records which are likely to refer to the same spoofed call. Once this is done other heuristics and analysis can be carried out as it is unlikely to be wasting computation on pairs that cannot be the same call due to differences in starting and ending times.

This results in two stages of record linkage. First, rule-based deterministic record linkage can be applied to determine if two records refer to the same call. This takes form in the timing-based heuristic which can be used to identify calls that require further analysis in the second stage. Probabilistic record linkage can losen the certainty that two candidate records refer to the same entity by assigning a likelihood based on further analysis such as fuzzy matching [17, p 955-958]

2.3.1 Timing Heuristic

Timing Analysis is a method of identifying that two separate records refer to the same entity when the payload is obscured by looking at timing meta-data.

The heuristic used to identify correlated calls is to analysis the timing data along with the type of call. This is because regardless of the mechanism used for spoofing, the ending times of the calls are expected to be within a zero to three second window. The type of call can either be Mobile Originated (MO) or Mobile Terminated (MT), dependent on this it is possible to determine which mechanism was used and hence decide which time frame the starting times are expected to be within.

In the case that the call types are MO and MT, this would indicate that either call redirection has been used where the record with MO is the spoofer, or that a spoofer is being called through the use of an access code that has been given to a non-spoofer. In either case the starting times are expected to be within zero to three seconds of each other like normal calls.

In the case that the call types are both MT, this would indicate that roaming callback mechanism has been used. When this is the case the difference between starting times should be between ten to fifteen seconds due to the fact the gang's Voice Over IP (VoIP) gateway has to call both participants which takes time.

Finally if the call types are both MO, the pair can be discarded as it is not possible to be the same call at this current moment.

2.3.2 Fuzzy Matching

Once the pairs of records have been identified from the timing analysis one can start looking for other patterns in the correlated calls. For example there can be patterns in the numbers that have been spoofed, although in theory these numbers can be randomised every call and have no pattern to extract. It has been observed from live data that often the spoofed number placed in the CLID by the VoIP gateway doesn't change with every call, similarly the access numbers given out to people who wish to contact a spoofer are not rotated enough.

Knowing this it is possible to check how similar the spoofed numbers are between candidate calls using various string comparison metrics. A few examples are the Levenshtein Distance or the Hamming Distance which both measure the dissimilarities between strings [18, Sec. 3, p 37]. If the similaritiy between the spoofed numbers is high a period of time then this adds reason to believe that the two people are in contact as the records

now match on not only timing analysis. There is now evidence to suggest that the numbers are not being changed often enough or that only some of the last digits are being changed to appear random.

2.4 Analysis

Once the tool can identify pairs of correlated records, analysis can be carried out to provide the user with meaningful results as opposed to just returning all the correlated calls which would quickly give no information other than the fact calls have occurred as the size increases. The benefits of analysing the results can be useful in identifying patterns and underlying structure.

2.4.1 Simple Analysis

A novel approach to analyse the results would be to simply give various statistics about the correlation between two handsets, for example how often we believe the call redirection mechanism to be used or the roaming callback mechanism along with the the proportion of these calls in the CDRs.

Once we have the information on all the correlated calls we can provide standard CDR analysis such as analysis on how often calls are made and the times handsets often call each other. The location of the handsets can be obtained by looking at the original records for the other attributes that were not needed for the tool to work out if calls could be correlated.

Metrics specific to spoofing can be calculated such as how often certain classes of numbers are used as the spoofed number. In the case of roaming callback the average time it takes for the infrastructure to setup the call could be measured by looking at the difference in starting times between handsets. In the case of call redirection it is possible to detect who initiated the call.

2.4.2 Social Network Analysis

More advanced analysis can be carried out once we have information for multiple different handsets as it is will be possible to analyse the correlated calls between all the various handsets. This can be thought of as an undirected weighted graph where the nodes are all the handsets and the weights are the number of times that two handsets have contacted each other.

This is known as Social Network Analysis [19] which is a fairly new field of research. This can be either visualised as a graph as mentioned above or the important information can displayed in a table. There are many different centrality measures which can provide information on the importance of nodes and edges in the network such as:

Degree Centrality Measurement of a node by the number of direct relations for a node in the graph.

Betweenness Centrality Measurement of a node by how many shortest paths pass through the node.

Closeness Centrality Measurement of a node by how close the it is to others in the network.

Eigenvector Centrality Measurement of a node by how much influence the node has.

2.5 Summary

To conclude the Literature Review a number of topics have been reviewed which are relevant to the project as both background information to aid understanding. And as methods that can be used to implement the objectives of the project.

First, key concepts have been introduced on how organised crime gangs are able to prevent previously established methods of linking calls between suspects through the use of USSD, CAMEL, and feature phones.

Research was then carried out to identify how to pre-process the data from the four major UK mobile network providers using semantic analysis into a standardised, concise, consistent format. This also provides a definition so that the tool can be expanded to more providers when required.

After this, the research was carried out on record linkage methods using the timing data and once found, looking for deeper patterns in the spoofed numbers themselves by measuring the similarity between strings.

Finally the review looked at various ways to analysis the data, simple methods such as primarily just displaying all information that may be useful and calculating a few statistics. While more advanced analysis methods have been looked at such as Social Network Analysis and how to apply it to the problem at hand.

3 Design and Methodology

3.1 Problem Statement and Analysis

Problem Statement: Given two sets of call data records (CDRs) that have different data structures and can have an arbitrary number of records as input; create an algorithm which can identify all the records from these CDRs which are the same call with obscurification applied. Furthermore the results and various metrics should be displayed to the user and saved for clerical review later if required.

One of the challenges of creating this tool will be parsing the different formats that the different mobile providers use such as converting between time zones, selecting the correct fields for called/calling numbers, and the practicality of dealing with a .xlsx or .csv file; potentially without knowing which provider the records come from.

The main challenge will be the generation of test data which can be used to verify the correctness of the program. The datasets generated will need to represent live data sufficiently and have the ability to change the proportions of correlated pairs and size to be able to test a wide range of input. Ideally the data will also be able to reflect the fact that the spoofed numbers are not changed regularly enough so that the tool's fuzzy matching heuristic can be tested.

A successful tool will be usable by experienced telecommunications analysis in law enforcement and will display all the information they require. Such as the records that have been identified using timing analysis, any metrics that can be calculated and finally if any deeper patterns can be found within the spoofed numbers themselves.

Additional information can be shown for individual records such as if call redirection or roaming callback was used; by looking at the call types and if any CAMEL data was present. This could be summed up to provide an numerical value of how often each mechanism is used in the suspected calls. If multiple people are entered the tool should be able to give this output for every pair of CDRs, this can then be used to build up a potential social graph. Graph centrality measures could also be displayed and highlight any central points of failure in the network.

3.2 Development Methodology

3.3 Requirements

3.3.1 Basic Requirements

The basic requirements are fundamental to the tool being usable.

- Parse the four main UK network providers' CDR formats into a standardised form.
- Return all the pairs of correlated calls that can be found using timing analysis.
- Provide feedback on how often each method was used to obscure calls.

3.3.2 Additional Requirements

Additional requirements would be ideal but are not necessary for the tool to be usable.

- Have a clear visual display of the information with the ability to search through the results.
- Be able to accept two or more sets of CDRs and check for correlated calls between all CDRs.
- Provide graph centrality measures if more than two CDRs have been given.
- Process other data of suspected calls, such as displaying on a map the location of the call using the fields from the original records.

3.3.3 Constraints

The main constraint of the project will be generating test data which is representational of live data that it can confidently be used to verify the correctness of the program. Furthermore there is a time constraint as the project needs to be delivered by April 30th.

3.4 Algorithmic Design

The tool's architecture is split up into three main modules; the parser, the correlator, and the data generation.

The parser and the correlator will make up a pipeline which takes raw CDRs to be standardised into a format that is understood by the correlator. This can then output the final results of the pairs that have been identified and any metrics that can be calculated and presented to the user and saved for inspection later if required.

The data generation module will be a stand-alone script that can take in various parameters such as the size of the CDRs and the proportion of irrelevant calls. The remainder will be split proportionally over using call redirection and roaming callback. In the case that one of the CDRs belongs to a normal SIM a proportion will also be supplied for how often the non-spoofer calls the spoofer. Although to the timing heuristic this is indistinguishable from call redirection. As this module will only be used to verify the correlator is working as intended, it will generate CDRs in the standardised format.

3.4.1 The Parser

The parser's job is to standardise any CDR from the main UK MNOs as each network has their own format. Firstly the algorithm needs to identify the network provider for a given CDR. This can be identified using the headers of the CDR as these change between providers. If this is not possible the algorithm should inform the user it was not possible to detect the network provider which is required to standardise fields such as the date-time field. Once it is known which provider the CDR is from, the algorithm can then create a standardised version of the CDR by removing the unnecessary attributes and translating the formats and units.

There are six fields that are potentially useful for the correlator, below is a table of these fields and the respective format to be converted to:

Field Name	Format/Type
ID	Integer
Start Date & Time	UTC
End Date & Time	UTC
Duration	Seconds
Call Type	MO/MT
Calling Number	11 Digits Without Country Code
Called Number	11 Digits Without Country Code

Table 3.1: Standardised CDR Data Types

3.4.2 The Data Generator

The data generator module will be build up from fundamental blocks which can be put together to generate an arbitrary CDR given parameters. The architecture will be a three layered design: The lowest layer of functions generates numbers on the fly, the second layer of functions create pairs of records which can be added to the CDR being generated. Finally the top layer creates a CDR given parameters using the pairs generated.

The lowest level of functions is those that generate numbers. There are three different functions on this level.

GeographicalNumber Generates a random UK landline number.

MobileNumber Generates a random UK mobile number.

FakeNumber Generates a random number which could either be: Mobile Number, Geographical Number, Area Codes That Don't Exist, Lazy Numbers, Wrong Length

The second layer of functions is those that generate pairs of records. There are six different types of calls that could be generated. These functions have also been divided to create the pair without the start and ending times, and then later on times can be assigned once all the pairs have been randomly shuffled in the CDR.

Spoofernon-spooferCR1 Pair of correlated records where a spoofer has called a non-spoofer using the call redirection mechanism.

Spoofernon-spooferRC1 Pair of correlated records where a spoofer has called a non-spoofer using the roaming callback mechanism.

non-spooferSpoofer1 Pair of correlated records where a non-spoofer has called a spoofer using an access code.

SpooferSpooferCR1 Pair of correlated records where a spoofer has called a spoofer using the call redirection mechanism.

SpooferSpooferRC1 Pair of correlated records where a spoofer has called a spoofer using the roaming callback mechanism.

non-spoofernon-spoofer1 Pair of records which are just a normal call. Only one record is used for the non-spoofer having a CDR generated.

The final layer has two functions: One that generates a pair of CDRs for a spoofer and a non-spoofer, and one that generates a pair of CDRs for two spoofers. Both of these functions take as input: Size, Minimum gap between calls, and the proportional of each type of correlated call.

3.4.3 The Correlator

The correlator loads in the standardised format that the parser gives out for the two CDRs that needs to be compared. The algorithm will compare all records with each other up to the point when they can be rejected, by filtering out the useless records it is possible reduce computation time.

First, every record in the first CDR is compared with every record in the second CDR on the end date-time field. If the end date-times are within zero to three seconds of each other then the next step is to look at the call types for each record to further narrow down the mechanism that has been used.

If both Call Types are Mobile Terminating (MT) then the records are the same call using the roaming callback method, or it's by chance these two numbers received a call ending at similar times. There is a distinction between these options, in the case of roaming callback mechanism being used, it is expected that the starting date-time of both calls are within a ten to fifteen second window. This window can be possible to edit if it is believed that the time it takes for the mechanism to work has changed. At this point the timing-based heuristic has identified a pair of correlated records which can be be analysed once all the pairs are found.

If one of the Call Types is Mobile Terminating (MT) while the other is Mobile Originating (MO) then these records are either the same call using the call redirection mechanism, or it's by chance these two unrelated calls ended at similar times. To further filter out records the starting time is expected to be within a zero to three second window of each other. If this is the case then the timing-based heuristic has identified a pair of correlated records which can be analysed further once all the pairs are found.

If the call types of both is Mobile Originating then the pair is discarded as they cannot be the same call using the mechanism known right now.

Algorithm 1 Timing-based heuristic

```
results = [] {2D array. Array containing arrays of (Probability,ID1,ID2)}
for all record1 in CDR1 do
    hits = BinarySearchEnddate-time(CDR2, 3) {Search for matching End date-time
    within 3 seconds}
    if hits == Null then
        Pass {No calls to check for this record}
    else
        for all record2 in hits do
            if record1.CallType == MT then
                if record2.CallType == MT then
                    startDiff = DurationCheck(record1.start,record2.start)
                    if durationDiff ≤ 10 or durationDiff ≥ 15 then
                        Pass
                    else
                        mechanism = RoamingCallback
                        results.append(record1, record2)
                    end if
                else
                    {record2.CallType == MO}
                    if record1.CallingNumber == record2.CalledNumber then
                        results.append(record1, record2) {Normal call without spoofing}
                    else
                        mechanism = CallRedirection
                        results.append(record1, record2)
                    end if
                end if
            else
                {record1.CallType == MO}
                if record2.CallType == MT then
                    if record1.CalledNumber == record2.CallingNumber then
                        results.append(record1, record2 {Normal call without spoofing}
                    else
                        mechanism = CallRedirection
                        results.append(record1, record2)
                    end if
                else
                    {record2.CallType == MO}
                    Pass {Not accounted for}
                end if
            end if
        end for
    end if
end for
```

3.5 Testing Design

There are three components that need to be tested to verify the correctness of the tool.

First, tests need to be carried out to ensure that the tool can convert at

least the four UK Mobile Network Providers into the standardised form, as the output from the parser will be used as input to find the correlated calls.

Second, the data generation module needs to be tested to verify that the input data used to test the tool is as expected and that all the correlated calls are labelled correctly so it is possible to verify the correctness of the tool.

Lastly, the tool that finds the correlated pairs using heuristics needs to be verified that all types of calls can be identified. Furthermore tests need to be carried out on the analysis that the tool carries out to ensure the correct metrics are presented to the user.

3.5.1 Testing The Parser

To verify the correctness of the parser, a white-box approach will be taken. The justification for this is that the format of the CDRs from the MNOs are known and can be verified by hand. Below is a table of the inputs that will be tested, which ensures that for these expected CDR formats, a standardised CDR can be generated.

Table 3.2: Input Format Tests

Provider	Date Format	Time Format
EE	dd/mm/yyyy	hh:mm:ss
H3G	dd/mm/yyyy	hh:mm:ss
H3G	dd/mm/yyyy	hh:mm
O2	dd/mm/yyyy	hh:mm:ss
Vodafone	dd/mm/yyyy	hh:mm:ss
Vodafone	dd/mm/yyyy	hh:mm

3.5.2 Testing The Data Generator

To verify the correctness of the data being generated, both white-box and black-box testing will be taken. This was decided upon as the method to generate this dataset is built up from many different functions. Hence white-box testing shall be used on the individual functions used to generate a CDR and black-box testing will be used to verify that these functions are being utilised correctly when generating larger CDRs.

I will also have to ensure the correctness of the data that is being generated as input. This will require unit tests on all the low level functions used to build up a CDR and test that the number of correlated calls produced in a

CDR is correct to the proportion supplied as a parameter. This primarily is testing that the output is correct regardless of if both numbers, one number, or no numbers are provided.

To carry out these tests unit tests shall be used for all the functions in the module, this includes testing the building block functions providing the various different values for parameters. This also includes testing the function which generates a set of data from various combinations of parameters provided below. It should be noted that the proportions are calculated from the remaining amount, for example if 75% is allocated to normal calls it is the decided what proportion of the remainder is used for call redirection and so on.

Table 3.3: Data Generation Parameters Tested

Parameter	Values Tested
Size	10,100,1000
Minimum Gap Between Calls	15,30,45,60
Proportion of Normal Calls	0%, 25%, 50%, 75%
Proportion of Call Redirection	0%, 25%, 50%, 75%
Proportion of Roaming Callback	0%, 25%, 50%, 75%
Proportion of non-spoofer Calling Spoofer	0%, 25%, 50%, 75%

3.5.3 Testing The Correlator

Next the tool itself will need to be verified that the behaviour is as expected, I have decided to do both white-box and black-box testing for this. In terms of white-box testing it is required to check that the functions which compare date-times are working as intended and that the control flow is correct.

For black-box testing I will be providing many different pairs of CDRs as input and use unit tests assert that the correct pairs of records are identified. I decided upon having many small tests which contain either zero, one or all records being of a certain type such as there only being one correlated call which used roaming callback. This will provide certainty at the edge cases that the calls are still identified regardless of how many there are. Furthermore I will enumerate many different synthetic datasets all with various sizes and proportions of correlated calls, by doing this I can also show that the tool can scale to larger input sizes and test many different proportions at many different sizes. Below is a list of the small tests which will look at edges cases and be small enough that manual review can further confirm that calls can be identified.

All Normal There are no pairs to be identified.

One Call Redirection There is just one pair to be identified that used call redirection.

One Roaming Callback There is just one pair to be identified that used roaming callback.

One non-spoofer Calling Spoofer There is just one pair to be identified which will be identified as call redirection.

All Call Redirection All pairs are to be identified using call redirection.

All Roaming Callback All pairs are to be identified using roaming callback.

All non-spoofer Calling Spoofer All pairs are to be identified as call redirection.

3.6 Implementation

3.6.1 Development Environment

I will be developing a CLI program in Python 3.7.1 in Ubuntu 18.10 64bit. All the libraries used in the implement come pre-installed with Python.

3.6.2 The Parser

The Parser is a python program which takes in csv files with the assumption that date-times are stored as follows:

EE dd/mm/yyyy hh:mm:ss

H3G dd/mm/yyyy hh:mm (or hh:mm:ss)

Vodafone dd/mm/yy hh:mm (or hh:mm:ss)

O2 dd/mm/yyyy hh:mm:ss

To test this I took a manual approach of supplying records with the format of each network provider along with sometimes including a country code that needs to be removed and converted.

3.6.3 The Correlator

The Correlator is a python program which takes as input two csv files of the standardised format and saves all the correlated pairs it can find into a new csv file. Furthermore the program also outputs the proportions that are correlated and which mechanism was used. The implementation closely follow the algorithm previously given in algorithm 1 on page 16 and a listing of the code is included in the Appendix B.1.

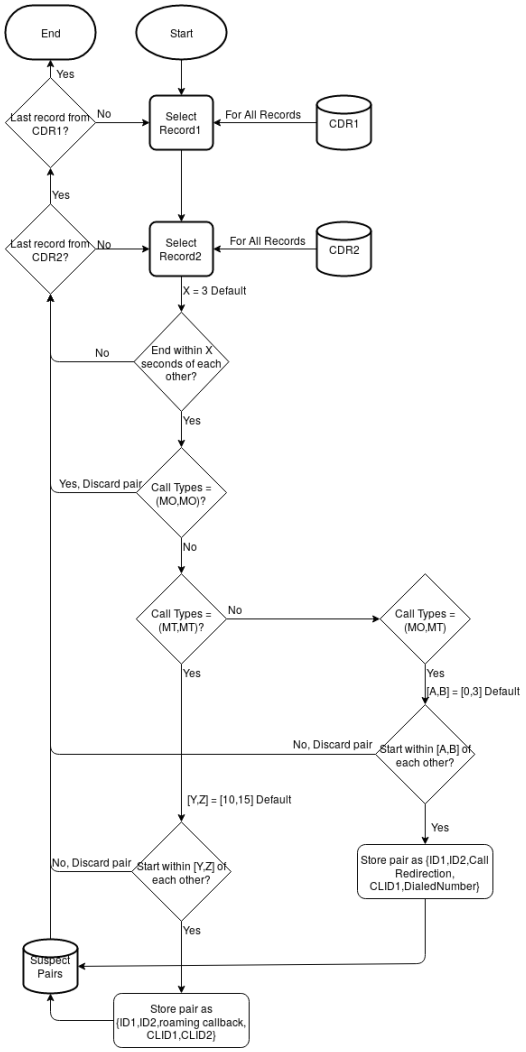

Figure 3.1: Implementation of the Timing Heuristic

3.6.4 Data Generator

The data generator is a python program which can generate pairs of CDRs with a specified proportion being correlated calls, along with a specified

size and the minimum gaps between calls. This is build up by firstly only generating the call types and numbers for a pair of records which can be either irrelevant filler calls or correlated calls of either roaming callback, call redirection, or a non-spoofer contacting a spoofer. Once the correct number of these half records have been created the program then randomly shuffles the calls and then assigns timestamps to them in order, making sure if it is a correlated call that the times are within the window expected. Finally IDs are attached for each record in a CDR and a class label is attached to indicate if the call should be identified and which mechanism was used.

The whole module can be iterated over if one wishes to generate large amounts of data for testing.

3.7 Summary

The design for the tool can be seen as three distinct parts; the parser, the correlator, and the data generator. The parser will be used to transform a CDR from any of the main telecommunication providers in the UK into a standardised form ready to be analysed. Once parsed, the correlator can apply algorithms to detect any potential correlated calls between two people using their normalised CDRs and provide an informational output of all the suspect calls and the proportions that were identified to be correlated calls. These scripts can be expanded on to compare many different pairs of CDRs sequentially.

4 Results and Evaluation

4.1 Results

My results for the unit tests for the correlator and the data generator confirmed that the data being generated was correct and that the correlator's internal functions are working as intended. When I tested the tool using the data generated, the tool managed to identify all correlated records regardless if roaming callback, call redirection or if it was a non-spoofer calling a spoofer. Moreover, changing the minimal gap between calls did not affect the results, this is a positive results as the tool was required to identify pairs regardless of where they are placed in relation to other calls.

The time taken to find all correlated calls increases rapidly due to every record at least comparing the end date-time with every other record. However even with this naive search, CDRs of size ten thousand could be completed in under five minutes on a four core laptop at 2.30Ghz.

4.2 Evaluation

The implication of these results is that as long as the input is in a standardised format ready for the correlator, then it is possible to automate the process of finding correlated calls which attempt to hide by spoofing the numbers in the records. However, unfortunately the tool does not implement fuzzy matching on the spoofed numbers after the timing analysis which could further increase the belief that calls are truly correlated. Furthermore when using the tool it currently has to be considered that the pairs identified have a slim chance of being unrelated calls which happen to start and end within the correct time-frames, although with larger input this should not reduce the confidence in the tool as if hundred pairs are identified the fact that there is a small chance they are not correlated is negligible.

4.3 Summary

In short, the results show that it is possible to carry out timing analysis on CDRs of stealth SIMs to identify calls between two handsets where at least one is using spoofing mechanisms to hide the numbers placed in CDRs.

5 Conclusion

In conclusion this project has shown that it is possible to automate the timing-based heuristic previously developed by Angus Marshall and Peter Miller [3] to identify calls that have been obfuscated with either call redirection or roaming callback.

5.1 Future Work

There are many areas where the tool can be further improved and built upon. Firstly the tool could be improved with testing on live data as the data generated does not model all the features of live data. For example the gaps between calls are always, at minimum, a certain amount plus some random seconds. This does not represent real data as it is likely that there will be peak times and off peak times. However the correlator doesn't look at the gaps and therefore can find pairs regardless of this fact. Furthermore, there is a uniform distribution on which type of numbers the fake numbers take, however this may not be representational of the live data as the spoofers might use innocent third party numbers more often than unallocated numbers.

A partial implementation was made for comparing the similarities between spoofed numbers however the data generation does not take parameters to simulate how often the spoofed numbers are changed and hence could not be tested if completed. This would also open up the possibility of assigning probabilities to a pair of calls instead of either discarding or identifying as it currently stands.

A small extension could be added on top of the program that allows the user to queue up pairs of CDRs to be investigated. This would be also be a step in the direction of allowing more than two CDRs to be submitted and calculating the graph centrality measures between all gang members under investigation.

Another adaptation would be to have the tool automatically adjust the time frames used to decide if a pair should be discarded which would allow the tool to start with small time frames. This would reduce the chance of catching false matches and then slowly increase the window until a significant number of calls are found without increasing the windows

beyond what is required.

Expanding the tool to be able to take in other prior knowledge such as if the phone is a feature phone (seized by law enforcement) could be used to further increase the probability that two calls are linked.

Finally it should also be considered that organised crime and law enforcement is a constant cat and mouse game. It seems feasible that the gangs' infrastructure could adapt to try and randomise the time between starting times for the roaming callback mechanism. Furthermore it may be possible to have two handsets both call into the VoIP and connect the call resulting in call types MO,MO being possible which the tool currently discards.

5.2 Closing Remarks

The results of this paper has shown that while organised crime continues to adopt stealth SIMs to obscure data often used by law enforcement, it is still possible to identify calls with heurisitics and analysis that is no longer solely looking for matching phone numbers. In the future more flexibility will be required as organised crime continues to adapt and attempts to further obscure and hide any patterns that help lead to detection.

A Appendix A

A.1 Glossary And Abbreviations

CAMEL Customised Applications for Mobile networks Enhanced Logic.

CDR Call Data Records.

CLID Calling Line Identity.

HLR Home Location Register.

HPLMN Home Public Land Mobile Network.

IMEI International Mobile Equipment Identifier.

IMSI International Mobile Subscriber Identifier.

Lazy Number A clearly fake number such as: 0777777777.

MNO Mobile Network Operator.

MO Mobile Originated.

MS Mobile Station.

MSC Mobile Switching Centre.

MT Mobile Terminated.

PLMN Public Land Mobile Network

SIM Subscriber Identity Module.

SIP Session Initiation Protocol.

USSD Unstructured Supplementary Service Data.

VLR Visitor Location Register.

VoIP Voice over IP.

VMNO Virtual Mobile Network Operator.

B Appendix B

B.1 The Correlator

```
1  import csv,datetime,sys
2
3  #CONST
4  dateTimeFormat = "%Y%m%d%H%M%S"
5  endDateTimeWindow = 3
6  callRedirectionWindow = [0,3]
7  roamingCallbackWindow = [10,20]
8  thresholdCLID = 5
9  thresholdAccessCode = 10
10
11 def SaveCDR(data,name):
12     with open(name+".csv","w",newline="") as csvfile:
13         writer = csv.writer(csvfile, delimiter=",")
14         writer.writerows(data)
15
16 def ReadCDR(filename):
17     CDR = []
18     with open(filename+".csv","r",newline="") as csvfile:
19         reader = csv.reader(csvfile,delimiter=",")
20         for record in reader:
21             CDR.append(record)
22     return CDR
23
24 def FilterEndDateTime(endDateTime1, endDateTime2):
25     if endDateTime1 > endDateTime2:
26         difference = endDateTime1 - endDateTime2
27     else:
28         difference = endDateTime2 - endDateTime1
29     if difference.seconds <= endDateTimeWindow:
30         return True
31     else:
32         return False
33
34 def GetCallType(type1,type2):
35     if (type1 == "MO") and (type2 == "MO"):
36         return False
37     elif (type1 == "MO") and (type2 == "MT"):
38         return "CR"
39     elif (type1 == "MT") and (type2 == "MO"):
40         return "CR"
41     elif (type1 == "MT") and (type2 == "MT"):
42         return "RC"
```

```
43      else:
44          raise ValueError("Call type(s) are not valid")
45
46  def FilterStartDateTime(mechanism,startDateTime1,startDateTime2)
        :
47      if startDateTime1 > startDateTime2:
48          difference = startDateTime1 - startDateTime2
49      else:
50          difference = startDateTime2 - startDateTime1
51      if mechanism == "RC":
52          if (difference.seconds >= roamingCallbackWindow[0]) and
                (difference.seconds <= roamingCallbackWindow[1]):
53              return True
54          else:
55              return False
56      elif mechanism == "CR":
57          if (difference.seconds >= callRedirectionWindow[0]) and
                (difference.seconds <= callRedirectionWindow[1]):
58              return True
59          else:
60              return False
61      else:
62          raise ValueError("Mechanism not valid")
63
64  def TimingHeuristicFilter(CDR1,CDR2):
65      pairs = []
66      for record1 in CDR1:
67          for record2 in CDR2:
68              endDateTime1 = datetime.datetime.strptime(record1
                    [2], dateTimeFormat)
69              endDateTime2 = datetime.datetime.strptime(record2
                    [2], dateTimeFormat)
70              if FilterEndDateTime(endDateTime1,endDateTime2):
71                  mechanism = GetCallType(record1[3],record2[3])
72                  if mechanism in ["CR","RC"]:
73                      startDateTime1 = datetime.datetime.strptime(
                            record1[1], dateTimeFormat)
74                      startDateTime2 = datetime.datetime.strptime(
                            record2[1], dateTimeFormat)
75                      if FilterStartDateTime(mechanism,
                            startDateTime1,startDateTime2):
76                          pairs.append((record1,record2,mechanism)
                                )
77      return pairs
78
79  def main():
80      filename1 = sys.argv[1]
81      if filename1[-4:] == ".csv":
82          filename1 = filename1[:-4]
83      filename2 = sys.argv[2]
84      if filename2[-4:] == ".csv":
85          filename2 = filename2[:-4]
86      CDR1 = ReadCDR(filename1)
87      CDR2 = ReadCDR(filename2)
88      timingResults = TimingHeuristicFilter(CDR1,CDR2)
```

```
89      numCR = 0
90      numRC = 0
91      results = []
92      ID = 0
93      for r in timingResults:
94          x = [str(ID)] + r[0] + r[1]
95          print(x)
96          ID += 1
97          results.append(x)
98          if r[2] == "CR":
99              numCR += 1
100         elif r[2] == "RC":
101             numRC += 1
102         else:
103             raise ValueError("Mechanism␣not␣valid␣after␣timing␣
                    heuistic␣filter")
104     numCorrelated = numCR + numRC
105     propCR = numCR / max(len(CDR1),len(CDR2))
106     propRC = numRC / max(len(CDR1),len(CDR2))
107     propCorrelated = numCorrelated / max(len(CDR1),len(CDR2))
108     print("The␣number␣of␣correlated␣calls␣using␣Call␣Redirection
            ␣found␣was:␣{0}".format(numCR))
109     print("And␣as␣a␣proportion␣this␣was:␣{0}".format(propCR))
110     print("The␣number␣of␣correlated␣calls␣using␣Roaming␣Callback
            ␣found␣was:␣{0}".format(numRC))
111     print("And␣as␣a␣proportion␣this␣was:␣{0}".format(propRC))
112     print("The␣number␣of␣correlated␣calls␣in␣total␣found␣was:␣
            {0}".format(numCorrelated))
113     print("And␣as␣a␣proportion␣␣this␣was:␣{0}".format(
            propCorrelated))
114     print("The␣records␣found␣have␣been␣saved␣in:␣{0}".format(
            filename1 + "RESULTS"))
115     SaveCDR(results,filename1+"RESULTS")
116
117 if __name__ == "__main__":
118     main()
```

B.2 The Correlator Test

```
1  import unittest,datetime,csv
2  import TheCorrelator as e
3
4  class TestTheCorrelator(unittest.TestCase):
5      def test_TimingHeuristicFilter(self):
6          inputs = []
7          for i in range(1,3):
8              CDR1 = []
9              with open(str(i)+"A.csv","r",newline="") as csvfile:
10                 reader = csv.reader(csvfile,delimiter=",")
11                 for record in reader:
12                     CDR1.append(record)
13             CDR2 = []
14             with open(str(i)+"B.csv","r",newline="") as csvfile:
```

```
15                  reader = csv.reader(csvfile,delimiter=",")
16                  for record in reader:
17                      CDR2.append(record)
18              inputs.append((CDR1,CDR2))
19          for CDRs in inputs:
20              CDR1 = CDRs[0]
21              CDR2 = CDRs[1]
22              timingResults = e.TimingHeuristicFilter(CDR1,CDR2)
23              trueCR = 0
24              trueRC = 0
25              for i in range(len(CDR1)):
26                  if CDR1[i][-1] == "1":
27                      trueCR += 1
28                  elif CDR1[i][-1] == "2":
29                      trueRC += 1
30                  elif CDR1[i][-1] == "3":
31                      trueCR += 1
32              numCR = 0
33              numRC = 0
34              for r in timingResults:
35                  if r[2] == "CR":
36                      numCR += 1
37                  elif r[2] == "RC":
38                      numRC += 1
39              for i in range(0,10):
40                  with self.subTest(i=i):
41                      if i == 0: #check number of RC is correct
42                          self.assertTrue(numRC == trueRC,msg="{0}
                                ␣|␣{1}␣|␣{2}".format(numRC,trueRC,len
                                (CDR1)))
43                      elif i == 1: #check number of RC/Access is
                            correct
44                          self.assertTrue(numCR == trueCR,msg="{0}
                                ␣|␣{1}␣|␣{2}".format(numCR,trueCR,len
                                (CDR2)))
45
46  if __name__ == "__main__":
47      unittest.main()
```

B.3 The Parser

```
1   import csv,datetime,sys
2
3   dateTimeFormat = "%Y%m%d%H%M%S"
4
5   def SaveCDR(data,name):
6       with open(name+".csv","w",newline="") as csvfile:
7           writer = csv.writer(csvfile,delimiter=",")
8           writer.writerows(data)
9
10  def ReadCDR(filename):
11      CDRRaw = []
12      with open(filename+".csv","r",newline="") as csvfile:
```

```
13          reader = csv.reader(csvfile, delimiter=",")
14          for record in reader:
15              CDRRaw.append(record)
16      return CDRRaw
17
18  def ParseO2CDR(CDRRaw):
19      ID = 0
20      CDR = []
21      for record in CDRRaw:
22          if record[0] not in ("MT","MO"):
23              continue
24          else:
25              #assign ID
26              r = []
27              r.append(ID)
28              ID += 1
29              #parse startDateTime
30              startDate = record[1] #dd/mm/yyyy
31              startDate = startDate[6:] + startDate[3:5] +
                    startDate[:2]
32              startTime = record[2] #hh:mm:ss
33              startTime = startTime[:2] + startTime[3:5] +
                    startTime[6:]
34              startDateTime = startDate + startTime
35              r.append(startDateTime)
36              #parse endDateTime
37              startDateTime = datetime.datetime.strptime(
                    startDateTime,dateTimeFormat)
38              duration = record[3] #hh:mm:ss
39              seconds = int(duration[6:])
40              minutes = int(duration[3:5])
41              hours = int(duration[:2])
42              duration = datetime.timedelta(seconds=seconds,
                    minutes=minutes,hours=hours)
43              endDateTime = startDateTime + duration
44              endDateTime = endDateTime.strftime(dateTimeFormat)
45              r.append(endDateTime)
46              #parse call type
47              callType = record[0]
48              r.append(callType)
49              #parse calling number
50              calling = record[6]
51              if calling[:3] == "+44":
52                  calling = "0" + calling[3:]
53              if (len(calling) != 10) and (len(calling) != 11):
54                  calling = calling.zfill(11)
55              r.append(calling)
56              #parse called number
57              called = record[10]
58              if called[:3] == "+44":
59                  called = "0" + called[3:]
60              if (len(called) != 10) and (len(called) != 11):
61                  called = called.zfill(11)
62              r.append(called)
63              CDR.append(r)
```

```
64      return CDR
65
66  def ParseVodafoneCDR(CDRRaw):
67      ID = 0
68      CDR = []
69      for record in CDRRaw:
70          if record[0] not in ("MT","MO"):
71              continue
72          else:
73              #assign ID
74              r = []
75              r.append(ID)
76              ID += 1
77              #parse startDateTime
78              startDateTime = record[2]
79              if len(startDateTime) == 19:
80                  startDateTime = startDateTime[6:10] +
                        startDateTime[3:5] + startDateTime[:2] +
                        startDateTime[11:13] + startDateTime[14:16] +
                         startDateTime[17:19]
81              elif len(startDateTime) == 16:
82                  startDateTime = startDateTime[6:10] +
                        startDateTime[3:5] + startDateTime[:2] +
                        startDateTime[11:13] + startDateTime[14:16] +
                         "00"
83              r.append(startDateTime)
84              #parse endDateTime
85              startDateTime = datetime.datetime.strptime(
                    startDateTime,dateTimeFormat)
86              duration = record[4]
87              duration = datetime.timedelta(seconds=int(duration))
88              endDateTime = startDateTime + duration
89              endDateTime = endDateTime.strftime(dateTimeFormat)
90              r.append(endDateTime)
91              #parse call type
92              callType = record[0]
93              r.append(callType)
94              #parse calling number
95              calling = record[5]
96              if calling[:3] == "+44":
97                  calling = "0" + calling[3:]
98              if (len(calling) != 10) and (len(calling) != 11):
99                  calling = calling.zfill(11)
100             r.append(calling)
101             #parse called number
102             called = record[6]
103             if called[:3] == "+44":
104                 called = "0" + called[3:]
105             if (len(called) != 10) and (len(called) != 11):
106                 called = called.zfill(11)
107             r.append(called)
108             CDR.append(r)
109     return CDR
110
111 def ParseEECDR(CDRRaw):
```

```
112     ID = 0
113     CDR = []
114     for record in CDRRaw:
115         if record[0] == "Start Date":
116             continue
117         else:
118             #assign ID
119             r = []
120             r.append(ID)
121             ID += 1
122             #parse startDateTime
123             startDate = record[0]
124             startTime = record[1]
125             startDateTime = startDate[6:] + startDate[3:5] +
                    startDate[:2] + startTime[:2] + startTime[3:5] +
                    startTime[6:8]
126             r.append(startDateTime)
127             #parse endDateTime
128             endDate = record[2]
129             endTime = record[3]
130             endDateTime = endDate[6:] + endDate[3:5] + endDate
                    [:2] + endTime[:2] + endTime[3:5] + endTime[6:8]
131             r.append(endDateTime)
132             #parse call type
133             callType = record[7]
134             r.append(callType)
135             #parse calling number
136             calling = record[4]
137             if calling[:3] == "+44":
138                 calling = "0" + calling[3:]
139             if (len(calling) != 10) and (len(calling) != 11):
140                 calling = calling.zfill(11)
141             r.append(calling)
142             #parse called number
143             called = record[5]
144             if called[:3] == "+44":
145                 called = "0" + called[3:]
146             if (len(called) != 10) and (len(called) != 11):
147                 called = called.zfill(11)
148             r.append(called)
149             CDR.append(r)
150     return CDR
151
152 def ParseH3GCDR(CDRRaw):
153     ID = 0
154     CDR = []
155     for record in CDRRaw:
156         if record[0] == "Date and Time of Call":
157             continue
158         else:
159             #assign ID
160             r = []
161             r.append(ID)
162             ID += 1
163             #parse startDateTime
```

```
164                 startDateTime = record[0]
165                 if len(startDateTime) == 19: #dd/mm/yyyy hh:mm:ss
166                     startDateTime = startDateTime[6:10] +
                            startDateTime[3:5] + startDateTime[:2] +
                            startDateTime[11:13] + startDateTime[14:16] +
                             startDateTime[17:19]
167                 elif len(startDateTime) == 16: #dd/mm/yyyy hh:mm
168                     startDateTime = startDateTime[6:10] +
                            startDateTime[3:5] + startDateTime[:2] +
                            startDateTime[11:13] + startDateTime[14:16] +
                             "00"
169                 else:
170                     raise ValueError("startDateTime␣not␣in␣correct␣
                            format")
171                 r.append(startDateTime)
172                 startDateTime = datetime.datetime.strptime(
                        startDateTime,dateTimeFormat)
173                 #parse endDateTime
174                 duration = record[8]
175                 duration = datetime.timedelta(seconds=int(duration))
176                 endDateTime = startDateTime + duration
177                 endDateTime = endDateTime.strftime(dateTimeFormat)
178                 r.append(endDateTime)
179                 #parse call type
180                 callType = record[7]
181                 r.append(callType)
182                 #parse calling number
183                 calling = record[1]
184                 if calling[:3] == "+44":
185                     calling = "0" + calling[3:]
186                 if (len(calling) != 10) and (len(calling) != 11):
187                     calling = calling.zfill(11)
188                 r.append(calling)
189                 #parse called number
190                 called = record[3]
191                 if called[:3] == "+44":
192                     called = "0" + called[3:]
193                 if (len(called) != 10) and (len(called) != 11):
194                     called = called.zfill(11)
195                 r.append(called)
196                 CDR.append(r)
197         return CDR
198
199     def MergeH3G(CDR1,CDR2):
200         #CDRs from H3G parser, no ID or class labels
201         CDR = []
202         t = datetime.datetime(2100,1,1)
203         index = -1
204         ID = 0
205         while (len(CDR1) != 0) or (len(CDR2) != 0):
206             #print("starting loop")
207             if len(CDR1) != 0:
208                 #print("CDR1 not empty, for loop time")
209                 for i in range(len(CDR1)):
210                     end = datetime.datetime.strptime(CDR1[i][2],
```

```
                    dateTimeFormat)
211                 if end < t:
212                     t = end
213                     index = (i,1)
214         if len(CDR2) != 0:
215             #print("CDR2 not empty, for loop time")
216             for i in range(len(CDR2)):
217                 end = datetime.datetime.strptime(CDR2[i][2],
                        dateTimeFormat)
218                 if end < t:
219                     t = end
220                     index = (i,2)
221         if index[1] == 1:
222             r = CDR1.pop(index[0])
223             CDR.append([ID] + r[1:])
224             ID += 1
225             t = datetime.datetime(2100,1,1)
226         else:
227             r = CDR2.pop(index[0])
228             CDR.append([ID] + r[1:])
229             ID += 1
230             t = datetime.datetime(2100,1,1)
231         #print(CDR)
232     return CDR
233
234 def IdentifyProvider(parameters):
235     if len(parameters) == 1:
236         filename = parameters[0]
237         if filename[-4:] == ".csv":
238             filename = filename[:-4]
239         CDR1 = ReadCDR(filename)
240         CDR2 = None
241     elif len(parameters) == 2:
242         filename = parameters[0]
243         if filename[-4:] == ".csv":
244             filename = filename[:-4]
245         filename2 = parameters[1]
246         if filename2[-4:] == ".csv":
247             filename2 = filename2[:-4]
248         CDR1 = ReadCDR(filename)
249         CDR2 = ReadCDR(filename2)
250     if CDR1[0][0] == "Start␣Date":
251         CDR = ParseEECDR(CDR1) #EE
252         SaveCDR(CDR, filename+"PARSED")
253     elif CDR1[0][0] == "Date␣and␣Time␣of␣Call" and CDR2[0][0] ==
            "Date␣and␣Time␣of␣Call":
254         CDRt1 = ParseH3GCDR(CDR1) #H3G
255         CDRt2 = ParseH3GCDR(CDR2)
256         CDR = MergeH3G(CDRt1,CDRt2)
257         SaveCDR(CDR, filename+"PARSED")
258     elif CDR1[0][0] == "Record":
259         CDR = ParseO2CDR(CDR1) #O2
260         SaveCDR(CDR, filename+"PARSED")
261     elif CDR1[0][0] == "Event␣-␣Tariff":
262         CDR = ParseVodafoneCDR(CDR1) #Vodafone
```

```
263             SaveCDR(CDR, filename+"PARSED")
264         else:
265             print("There␣is␣no␣headers␣to␣detect␣the␣mobile␣network"
                    )
266             print("1.␣EE")
267             print("2.␣H3G")
268             print("3.␣O2")
269             print("4.␣Vodafone")
270             x = int(input("Please␣select␣an␣option␣above:␣"))
271             if x == 1:
272                 CDR = ParseEECDR(CDR1)
273                 SaveCDR(CDR, filename+"PARSED")
274             elif x == 2:
275                 if len(parameters) != 2:
276                     raise ValueError("Two␣files␣are␣required␣for␣H3G
                            ,␣both␣incoming␣and␣outgoing")
277                 else:
278                     CDRt1 = ParseH3GCDR(CDR1)
279                     CDRt2 = ParseH3GCDR(CDR2)
280                     CDR = MergeH3G(CDRt1,CDRt2)
281                     SaveCDR(CDR, filename+"PARSED")
282             elif x == 3:
283                 CDR = ParseO2CDR(CDR1)
284                 SaveCDR(CDR, filename+"PARSED")
285             elif x == 4:
286                 CDR = ParseVodafoneCDR(CDR1)
287                 SaveCDR(CDR, filename+"PARSED")
288             else:
289                 raise ValueError("That␣is␣not␣a␣valid␣option!")
290
291 def main():
292     IdentifyProvider(sys.argv[1:3])
293
294 if __name__ == "__main__":
295     main()
```

Bibliography

[1] Ofcom. (2018). Telecommunications market data update q3 2018, [Online]. Available: https://www.ofcom.org.uk/research-and-data/telecoms-research/data-updates/telecommunications-market-data-update-q3-2018 (visited on 09/04/2019).

[2] OpenCellID. (2019). Statistics on cell tower numbers, [Online]. Available: https://opencellid.org/stats.php (visited on 28/04/2019).

[3] A. Marshall and P. Miller, 'Mobile phone call data obfuscation & techniques for call correlation', 2019.

[4] G. Interceptor. (Apr. 2019). Used - samsung s3 gsm interceptor edition, [Online]. Available: http://gsminterceptor.co.uk/product/used-samsung-s3-gsm-interceptor-edition/ (visited on 27/03/2019).

[5] Endoacustica. (Apr. 2019). Anti-tapping/anti-tracking mobile stealth phone plus, [Online]. Available: https://www.endoacustica.com/advanced-anti-tapping-tracking-mobile-stealth-phone-plus.html (visited on 27/03/2019).

[6] E. T. S. Institute. (Dec. 1996). Digital cellular telecommunications system; unstructured supplementary service data (ussd) - stage 2(gsm 03.90). version 5, [Online]. Available: https://www.etsi.org/deliver/etsi_gts/03/0390/05.00.00_60/gsmts_0390v050000p.pdf (visited on 09/04/2019).

[7] Wikipedia. (Dec. 2018). Unstructured supplementary service data, [Online]. Available: https://en.wikipedia.org/wiki/Unstructured_Supplementary_Service_Data (visited on 10/04/2019).

[8] E. T. S. Institute. (Oct. 2014). Digital cellular telecommunications system (phase 2+);universal mobile telecommunications system (umts); customised applications for mobile network enhanced logic (camel) phase x; camel application part (cap) specification (3gpp ts 29.078 version 12.0.0 release 12), [Online]. Available: https://www.etsi.org/deliver/etsi_ts/129000_129099/129078/12.00.00_60/ts_129078v120000p.pdf (visited on 09/04/2019).

[9] 3GPP. (Jun. 2018). Customised applications for mobile network enhanced logic (camel) phase 4; stage 2. version 15.0.0, [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=766 (visited on 10/04/2019).

[10] Wikipedia. (Sep. 2018). Intelligent network, [Online]. Available: https://en.wikipedia.org/wiki/Intelligent_Network (visited on 20/04/2019).

[11] Voipfone. (Apr. 2019). Voipfone hosted pbx, [Online]. Available: https://www.voipfone.co.uk/switchboard_and_hosted_PBX_services.php (visited on 09/04/2019).

[12] Wikipedia. (Apr. 2019). Session initiation protocol, [Online]. Available: https://en.wikipedia.org/wiki/Session_Initiation_Protocol (visited on 15/04/2019).

[13] ——, (Nov. 2018). A5/1, [Online]. Available: https://en.wikipedia.org/wiki/A5/1 (visited on 09/04/2019).

[14] G. Interceptor. (Apr. 2019). Number/voice changing sim cards, [Online]. Available: http://gsminterceptor.co.uk/product/number-changing-sim-cards/ (visited on 09/04/2019).

[15] P. Christen, *Data Matching: Concepts And Techniques For Record Linkage, Entity Resolution, and Duplicate Detection*. Springer, 2012, ISBN: 978-3-642-31164-2. DOI: 10.1007/978-3-642-31164-2_2. [Online]. Available: https://doi.org/10.1007/978-3-642-31164-2_2.

[16] T. Herzog, F. Scheuren and W. Winkler, *Data Quality And Record Linkage Techniques*. Springer, 2007.

[17] A. Sayers, Y. Ben-Shlomo, A. Blom and F. Steele, 'Probabilistic record linkage', 2017.

[18] G. Navarro, 'A guided tour to approximate string matching', *ACM Comput. Surv.*, vol. 33, no. 1, pp. 31–88, Mar. 2001, ISSN: 0360-0300. DOI: 10.1145/375360.375365. [Online]. Available: http://doi.acm.org/10.1145/375360.375365.

[19] K. Zweig, *Network Analysis Literacy*. Springer, 2016.