User Manual for leakiEst v.1.2 – a Tool for Estimating Information Leakage

Tom Chothia, Yusuke Kawamoto, and Chris Novakovic

School of Computer Science, University of Birmingham, Birmingham, UK

leakiEst estimates the min-entropy leakage and mutual information measures based on trial runs of a system. It operates on both discrete and continuous data and performs statistical tests to distinguish an insecure system with a very small information leak from a secure one with no leakage.

1 Usage of the Tool

leakiEst is available as a JAR file, `leakiest-1.2.jar`, from `http://www.cs.bham.ac.uk/research/projects/infotools/leakiest/`. It can be invoked from the command line as follows:

```
> java -jar leakiest-1.2.jar <options>
```

It supports reading options from a configuration file:

```
> java -jar leakiest-1.2.jar -cfg configuration.txt
```

The help option `-h` displays the other command-line options supported by the tool:

```
> java -jar leakiest-1.2.jar -h
```

2 Main Command-Line Options

We now explain the purpose of the main command line options. leakiEst can calculate four types of leakage measure: mutual information (`-mi`), capacity (`-cp`), min-entropy leakage (`-me`) and min-capacity (`-mc`).

```
-mi                  Calculate mutual information
-cp                  Calculate capacity
-me                  Calculate min-entropy leakage
-mc                  Calculate min-capacity
```

Note that the tool shows the confidence interval and clarifies whether or not there is evidence of information leakage when calculating mutual information or min-entropy leakage as the leakage measure.

Other options allow us to choose whether the input data is discrete or continuous (but only when calculating mutual information as the leakage measure):

```
-di                    Discrete data
-co                    Continuous data
```

The tool supports two observation file formats (for storing data observed in a system), a channel format (for describing a channel matrix), and Weka's ARFF format (as described in previous sections).

```
-o <file>              Read input from an observation file
-o2 <file1> <file2>    Read input from two observation files, each
                       recording the observation of one attribute
-c <file>              Read input from a channel file
-a <file>              Read input from an ARFF file
```

The `-v` option is used to change the amount of information displayed, and the `-p` option is used to print a channel matrix when discrete data is supplied to the tool.

```
-v <level>             Set the level of information shown (0 to 5)
                       e.g. -v 4
-p                     Print a channel matrix
```

3 Example 1: Dining Cryptographers Protocol

In our first example, we calculate the mutual information from the channel matrix of a Dining Cryptographers protocol with three participants and biased coins. To analyse the channel matrix, we edit the configuration file `configDC.txt` to specify the location of the channel file `dc3allbias4.txt` and run the following command:

```
> java -jar leakiest-1.2.jar -cfg configDC.txt
```

The tool gives the following output:

```
These observations lead to the following channel matrix, to 4 decimal places:
          | AAD    | ADA    | DAA    | DDD
 user1    | 0.1875 | 0.1875 | 0.4375 | 0.1875
 user2    | 0.1875 | 0.4375 | 0.1875 | 0.1875
 user3    | 0.4375 | 0.1875 | 0.1875 | 0.1875

Mutual information: 0.1038   Calculated with the uniform input distribution.
The attacker learns 0.1038 bits, out of a possible
1.585 bits, about the input events.
```

This result shows that the mutual information calculated from the channel matrix in the channel file and the uniform input distribution is 0.1038 bits.

If we replace the option `-mi` with `-cp`, the tool calculates the capacity of the channel as a leakage measure, as well as the input distribution that achieves the capacity (i.e., that maximises the mutual information):

```
> java -jar leakiest-1.2.jar -cp -c data/dc3allbias4.txt -v 2
```

```
Maximising input distribution estimated to be:
  [ user1: 0.3333, user2: 0.3333, user3: 0.3333 ]
  Capacity calculated to within acceptable error, in 1 iterations.
Capacity: 0.1038
The attacker learns 0.1038 bits, out of a possible
1.585 bits, about the input events.
```

The tool uses the Blahut-Arimoto algorithm to calculate capacity. We can specify the following two parameters used in the algorithm:

```
-e <level>          Set the acceptable error level for Blahut-
                    Arimoto algorithm when computing capacity
                    e.g. -e 0.0000001
-i <number>         Set the maximum number of iterations
                    e.g. -i 500
```

4 Example 2: Analysis of Encrypted Tor Traffic

To analyse encrypted Tor traffic data, we edit the configuration file `configTor.txt` to specify the location of the ARFF file `TorDataset.arff` and run the following command.

```
> java -jar leakiest-1.2.jar -cfg configTor.txt
```

We use the following two options in the configuration file (or from the command line if not using a configuration file):

```
-high <numbers>      specify indeces of high value features
                         e.g. -high 1,3,4
-low <numbers>       specify indeces of low value features
                         e.g. -low 12,13  -low @all
```

We set `-high 75` and `-low @each` in `configTor.txt`, so that the 76th attribute ("`class`") in the ARFF file is regarded as a secret (high attribute) and each of the other attributes is regarded as an observation (low attribute).

The tool gives the following output that ranks the estimated mutual information of each attribute (note that most of the output is omitted here for brevity):

```
Confidence Result    Attributions                                      Range(with ...
LEAK       1.431 for  6 payload_size_sent        zero leakage < 0.589  [1.264, 1.599]
LEAK       1.220 for  8 number_spike_feature     zero leakage < 0.431  [1.044, 1.395]
...
ZERO LEAK  0.839 for  1 number_packet_sent       zero leakage < 0.842  [0.670, 1.007]
...
```

This result shows that the attribute "`number_spike_feature`" leaks the most information from the attribute "`class`" among all the attributes. "`zero leakage < 0.179`" shows that there is no evidence of leakage if the estimated value is less than 0.179 bits. On the column furthest on right-hand side, the 95% confidence intervals for the mutual information are shown.

5 Example 3: Analysis of Malicious JavaScript

This dataset relates a binary value indicating the maliciousness of a particular piece of JavaScript code (the secret) to characteristics that can be inferred by observing or executing the code (the public outputs).

To analyse this dataset, we edit the configuration file `configJavaScript.txt` to specify the location of the ARFF file `JavaScriptFull.arff` and run the following command:

```
> java -jar leakiest-1.2.jar -cfg configJavaScript.txt
```

We set `-high 69` and `-low @each` in `configJavaScript.txt`, so that the 70th attribute ("`Intent`") in the ARFF file is regarded as a secret (high attribute) and each of the other attributes is regarded as an observation (low attribute).

Then we obtain the following result for the estimated mutual information:

```
Confidence Result    Attributions                                          Range(with...
LEAK       0.714 for 10 Ratio_Of_Definitions_To_Uses zero leakage < 0.262 [0.678, 0.750]
LEAK       0.647 for 6 Total_Bytes_Allocated         zero leakage < 0.412 [0.613, 0.680]
...
ZERO LEAK  0.397 for 68 Script_Size                  zero leakage < 0.853 [0.379, 0.416]
...
```

6 Example 4: Analysis of Biometric Passport Protocol

To analyse passport data, we edit the configuration file `configPassportGerman.txt` to specify the locations of the two observation data files for passport analysis, and run the following command:

```
> java -jar leakiest-1.2.jar -cfg configPassportGerman.txt
```

The tool produces the following output for the German passport data:

```
Estimated mutual information: 0.9822 (out of possible 1.000 bits)
There is a leak.
Estimate is NOT below 0.0253 (the 95 percentile for shuffled values).
```

This output shows that there is evidence of information leakage; the estimated mutual information of 0.9822 bits for the passport data is larger than the 95% confidence interval of mutual information in the case of zero information leakage.

If we use the passport data obtained by adding padded time delays to the original data, the tool produces the following output.

```
> java -jar leakiest-1.2.jar -cfg configPassportGermanFix.txt
```

```
Fixing the data of passport analyses...
Estimated mutual information: 0.1592 (out of possible 1.000 bits)
No leak detected.
Estimate is below 0.3184 (the 95 percentile for shuffled values).
```

This means that there is no evidence of information leakage; the estimated mutual information of 0.1592 bits for the passport data is smaller than the 95% confidence interval of mutual information in the case of zero information leakage.

7 Example 5: Analysis of Dining Cryptographers Protocol with TLS

This dataset is an observation file containing 10,000 trial runs of a variant of the Dining Cryptographers protocol whereby the secret payer sends his credit card number over an encrypted link before reporting the values of his coins; the public outputs are the values of the coins as announced by each participant, in the order in which they announce them. The possible source of leakage here is that in a broken implementation, the payer may take longer to announce the value of their coin, which may reveal the identity of the payer.

To analyse this dataset, we edit the configuration file `configTLS.txt` to specify the location of the observation file `DCprotocol_with_TLS_pc.txt` and run the following command:

```
> java -jar leakiest-1.2.jar -cfg configTLS.txt
```

We are given the following estimated min-entropy leakage (again, note that most of the output is omitted here for brevity):

```
Estimated min-entropy: 1.5611 (out of possible  1.585 bits)
  Calculated with the uniform input distribution.
  Possible error: 0.3393
  Between 1.2218 and 1.585 with 95% confidence
There is a leak.
```

In this result, the 95% confidence interval is calculated and used to decide whether or not there is evidence of leakage.