User Manual for leakiEst v.1.4 – a Tool for Estimating Information Leakage

Tom Chothia, Yusuke Kawamoto, and Chris Novakovic

Abstract. leakiEst [CKN13] estimates the mutual information and min-entropy leakage 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 v.1.4.n is available as a JAR file, `leakiest-1.4.`n`.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-1.4.n.jar <options>
```

It supports reading options from a configuration file:

```
> java -jar leakiest-1.4.n.jar -cfg configuration.txt
```

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

```
> java -jar leakiest-1.4.n.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
```

These calculations are based on the methods presented in [CCG10,CG11,CKN14]

The tool supports two observation file formats (for storing data observed in a system), a channel format (for describing a channel matrix), and Weka [HFH+09]'s ARFF format [wek].

```
-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
```

The `-hcomp` option can be used to see the usage of compositional reasoning [KCP14], which has not been maintained since version 1.3.

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-1.4.n.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.

We can also run this command without using the configuration file:

```
> java -jar leakiest-1.4.n.jar -mi -c data/dc3allbias4.txt -p -v 2
```

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-1.4.n.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 `Tor.arff` and run the following command.

```
> java -jar leakiest-1.4.n.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 indices of high value features
                            e.g. -high 1,3,4
  -low <numbers>        specify indices 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).

We can also run this command without using the configuration file:

```
> java -jar leakiest-1.4.n -mi -di -a data/Tor.arff -high 69 -low @each -v 0
```

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 [1.315, 1.547]
LEAK       1.220 for  8 number_spike_feature zero leakage < 0.431 [1.098, 1.341]
...
ZERO LEAK  0.245 for  34 payload_spike_6     zero leakage < 0.537 [0.109, 0.382]
...
```

This result shows that the attribute "`payload_size_sent`" leaks the most information from the attribute "`class`" among all the attributes. "`zero leakage` $<$ `0.589`" shows that there is no evidence of leakage if the empirical value (without bias correction) is less than 0.589 bits. On the column furthest on right-hand side, the 95% confidence intervals for the mutual information are shown.

In the zero leakage test, a confidence interval for the mutual information of a zero-leakage dataset obtained by destroying the link between the secrets and observables by shuffling as described in Section IV of [CG11].

Note that the result 1.431 of discrete mutual information has been corrected by removing its bias from the empirical value whereas the zero leakage test compares the empirical value (without bias correction) with 0.589.

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-1.4.n.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).

We can also run this command without using the configuration file:

```
> java -jar leakiest-1.4.n -mi -di -a data/JavaScriptFull.arff
                           -high 69 -low @each -v 0
```

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

```
Confidence Result    Attributions                                        Range(with...
LEAK       0.508 for 10 Ratio_Of_Definitions_To_Uses zero leakage < 0.262 [0.484, 0.531]
LEAK       0.459 for 16 Method_Call                  zero leakage < 0.007 [0.424, 0.494]
...
ZERO LEAK  0.147 for 68 Script_Size                  zero leakage < 0.853 [0.137, 0.157]
...
```

6 Example 4: Analysis of Biometric Passport Protocol

To analyse the e-passport data studied in [CS10], 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-1.4.n.jar -cfg configPassportGerman.txt
```

We can also run this command without using the configuration file:

```
> java -jar leakiest-1.4.n.jar -mi -co -v 0
                               -o2 data/timesGerman500Related.txt
                                   data/timesGerman500Random.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 0.0125(the 95 percentile for shuffled values).
```

This output shows that the estimated mutual information is 0.9822 bit, and that there is evidence of information leakage; the mutual information 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-1.4.n.jar -cfg configPassportGermanFix.txt
```

```
Estimated mutual information: 0.1524 (out of possible  1.000 bits)
No leak detected.
  Estimate is below 0.2744(the 95 percentile for shuffled values).
```

This means that there is no evidence of information leakage; the mutual information 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-1.4.n.jar -cfg configTLS.txt
```

We can also run this command without using the configuration file:

```
> java -jar leakiest-1.4.n -me -di -o data/DCprotocol_with_TLS_pc.txt
```

We are given the following estimated min-entropy leakage:

```
Estimated min-entropy leakage: 1.5555 (out of possible  1.579 bits)
  Possible error: 0.019
  Between 1.5365 and 1.5644 with 95.0% confidence
There is a leak.
```

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

References

CCG10. Konstantinos Chatzikokolakis, Tom Chothia, and Apratim Guha. Statistical Measurement of Information Leakage. In *Proc. TACAS*, pages 390–404, 2010.

CG11. Tom Chothia and Apratim Guha. A statistical test for information leaks using continuous mutual information. In *Proc. of CSF*, pages 177–190, 2011.

CKN13. Tom Chothia, Yusuke Kawamoto, and Chris Novakovic. A tool for estimating information leakage. In *Proc. of CAV 2013*, pages 690–695, 2013.

CKN14. Tom Chothia, Yusuke Kawamoto, and Chris Novakovic. LeakWatch: Estimating information leakage from java programs. In *Proc. of ESORICS 2014 Part II*, pages 219–236, 2014.

CS10. Tom Chothia and Vitaliy Smirnov. A traceability attack against e-passports. In *FC10: Proceedings of the 14th Int. Conf. on Financial Cryptography and Data Security 2010*. LNCS, Springer-Verlag, 2010.

HFH+09. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA Data Mining Software. *SIGKDD Explorations*, 11(1):10–18, 2009.

KCP14. Yusuke Kawamoto, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. Compositionality results for quantitative information flow. In *Proc. of QEST'14*, pages 368–383, 2014.

wek. Weka — ARFF. `http://weka.wikispaces.com/ARFF`.