An Interactive Tool for Learning Linear and Differential Cryptanalysis of SPNs
Mike Carpenter
<mjc7806@g.rit.edu>
# Table of Contents

- Introduction..................................................................................................................4
- Background....................................................................................................................4
  - Substitution-Permutation Networks........................................................................4
  - Linear Cryptanalysis.................................................................................................5
    - Linear Approximations.........................................................................................5
    - Subkey Extraction.................................................................................................5
  - Differential Cryptanalysis.........................................................................................6
    - Differential Approximations.................................................................................6
    - Subkey Extraction.................................................................................................6
- Literature Review..........................................................................................................7
  - Anane........................................................................................................................7
  - Fuzi...........................................................................................................................8
  - Hu..............................................................................................................................9
  - Picek........................................................................................................................10
  - Tao............................................................................................................................10
  - DiffCryptDemo.......................................................................................................11
- Evaluation Criteria.......................................................................................................12
  - Requirements..........................................................................................................13
  - Considerations.........................................................................................................13
- Project Design and Implementation............................................................................14
  - Overview..................................................................................................................14
  - External Libraries....................................................................................................14
  - Project Structure.....................................................................................................14
  - SPN Construction Structure....................................................................................15
  - LDC Structure..........................................................................................................15
  - UI Structure.............................................................................................................15
    - Master Properties Dialog.....................................................................................15
    - S-box Definition Dialog......................................................................................16
    - Permutation Definition Dialog............................................................................18
    - Cipher Selection Dialog......................................................................................19
    - SPN Definition Dialog........................................................................................20
    - SPN Visualization Frame......................................................................................21
    - Cryptanalysis Dialogs..........................................................................................22
- Project Evaluation........................................................................................................24
  - Evaluation of UI Design Criteria............................................................................24
    - New/Load and Master Properties Dialogs..........................................................24
    - SPN Definition Dialog.........................................................................................24
    - SPN Visualization Frame......................................................................................24
    - Component Definition Dialogs.............................................................................25
    - Cryptanalysis Dialogs..........................................................................................25
  - Retrieval of Target Partial Subkeys..........................................................................25
  - Example Report.........................................................................................................26
  - Development Roadblocks.........................................................................................28
  - SPN Drawing Issues.................................................................................................29
Introduction

Linear and differential cryptanalysis have existed for years as a set of tools to establish a metric of resiliency for cryptographic ciphers utilizing the substitution-permutation network (SPN) design, a category of cryptographic cipher in which many popular ciphers, such as the Advanced Encryption Standard (AES). This is typically an involved process with many manual steps, often written specifically for the SPN being analyzed, with minimal reusable or generalized code. As of this writing, there do not exist any tools to make the process interactive for an arbitrary SPN; the few interactive, automated tools that do exist are designed around a particular predefined cipher.

The goal of this project is to create Maledict\textsuperscript{[1]}, a Java application inspired and influenced by the tutorial by Howard Heys\textsuperscript{[4]}, with a user interface in Swing in order to provide an interactive experience in the linear and differential cryptanalysis of arbitrary, user-designed substitution-permutation networks. As a part of the interactive process I seek to provide visualizations of various features, such as the pseudo-linear “path” followed through the network by a given bit. I also intend to report meaningful results obtained using the tool, such as linear approximation/difference distribution tables or subkey biases from experimental results.

It is my intention that this project provide a foundation for an effective academic tool, which can be used by students to better understand and observe the processes of linear and differential cryptanalysis.

Background

Substitution-Permutation Networks

Substitution-permutation networks (SPNs) are a type of cryptographic block cipher and are a popular underlying encryption technology behind many conventional modern cryptographic ciphers, including Twofish and AES. Therefore, understanding SPNs and methods of cryptanalysis to test their viability is important knowledge in the modern computer security world.

In an SPN, the bits in a block pass through a number of rounds. Each round consists of: a subkey, a partition of the larger key equivalent to the block size, on which a reversible bitwise operation is performed on input data (in the case of this report, all keys will be applied using XOR); a set of substitution boxes (S-boxes), which accept some input and, using an input-output mapping, produce an output; and a permutation, wherein the input blocks are shuffled and reordered.

While an S-box does not necessarily have an equal number of input and output bits, this project only concerns itself with bijective S-boxes, which do. S-boxes are the major target of cryptanalysis attacks against SPNs, as there is no predetermined function to determine output from input other than a direct mapping—therefore, approximating certain properties about these S-boxes allow us to determine unknown aspects of the cipher, namely pieces of the key. The two cryptanalysis methods explored in
this project, both of which concern themselves with approximating S-boxes in this manner, are linear and differential cryptanalysis.

**Linear Cryptanalysis**

**Linear Approximations**

Linear cryptanalysis is a method of cryptanalysis wherein some of the input bits to an S-box are approximated to have a linear relation with some of the output bits to that same S-box. In essence, a line can be drawn through the S-box from one set of bits to another in a manner similar (though not perfectly analogous) to a permutation.

A linear approximation of an S-box differs from a permutation in that a given set consisting of one or more input bits may correlate to another set consisting of one or more output bits. That is, rather than an individual input bit having a one-to-one correspondence with a single output bit, as in a permutation, the relationship in a linear approximation is between two sets of bits. The equivalence of an S-box's input bits with its output bits is measured using an XOR operation. That is, given a 4-bit bijective S-box where bit 2 of the input (X) correlates with bits 2 and 3 of the output (Y):

\[
X_2 = Y_2 \oplus Y_3 \\
X_2 \oplus Y_2 \oplus Y_3 = 0
\]

An approximation is chosen based on constructing a linear approximation table, wherein a given approximation is compared against all possible inputs to the S-box. These approximations are chained together throughout the cipher, to form an approximation based on plaintext characters and the input bits to the final round of the cipher.

It is not necessary to consider the bits of keys in previous rounds, as in given approximation the set of key bits is constant, and those bits logically converge to a single constant bit value of 0 or 1.

**Subkey Extraction**

A portion of the last round's key corresponding to the bits exiting the relevant S-boxes in the final round (called the target partial subkey) can be extracted through this approximation: when given the full-cipher approximation, a large number of known plaintext/ciphertext pairs are partially-decrypted using all possible candidate keys (performing one round of decryption to obtain the input to the last round of S-boxes).

Each decryption is examined for conformity to the linear approximation, and that key is assigned a bias. In the case of linear cryptanalysis, a key's bias is the magnitude of the difference between chance (50%) and that approximation's conformity amongst the plaintext/ciphertext pairs for that given key. The key with the largest absolute bias is assumed to be the correct key.

Linear cryptanalysis doesn’t require knowledge of previous rounds’ keys, avoiding brute force. Brute-forcing the full cipher requires trying more than 1.2 septillion \(2^{80}\) keys. However, linear
cryptanalysis can obtain 8 bits of the last-round subkey through by testing testing 256 \(2^8\) subkeys against a set of known pairs. The subkeys of prior rounds are ultimately irrelevant.

**Differential Cryptanalysis**

**Differential Approximations**

A differential approximation, or just a differential, can be visualized in a similar way to a linear approximation (a line drawn through S-boxes from one set of bits to another), but has a fundamentally different meaning.

Contrary to a linear approximation, a differential approximation involves two inputs to an S-box and their corresponding outputs. Two input pairs are chosen with a given difference, and that difference (the result of an XOR operation between the two inputs) serves as the input to the differential. The corresponding outputs are then XORed together to determine the difference between them. For two given inputs \((A_i, B_i)\) and outputs \((X_i, Y_i)\) we can calculate an input differential \((C_i)\) and output differential \((Z_i)\):

\[
\begin{align*}
C_i &= A_i \oplus B_i \\
Z_i &= X_i \oplus Y_i
\end{align*}
\]

Then, the differential is tested similarly to the input/output masks for linear approximation. For a 4-bit S-box differential:

\[
\begin{align*}
C_0 \oplus C_1 \oplus C_2 \oplus C_3 &= Z_0 \oplus Z_1 \oplus Z_2 \oplus Z_3 \\
C_0 \oplus C_1 \oplus C_2 \oplus C_3 \oplus Z_0 \oplus Z_1 \oplus Z_2 \oplus Z_3 &= 0
\end{align*}
\]

The frequency of output differentials are stored in a difference distribution table, which is analogous to a linear approximation table. For all possible input pairs to the S-box, the outputs are calculated, and the cell intersecting the input difference and output difference is incremented for each occurrence.

Ultimately, those differentials which occur most often in the difference distribution are the most attractive approximations to use in a full-cipher differential, as they are most likely to occur among arbitrary chosen plaintext/ciphertext pairs.

In the case of differential cryptanalysis, the influence of the key on the approximation disappears entirely, and therefore the subkeys of prior rounds can be ignored. As both inputs to an S-box are XORed with the same key bits, the two copies of the subkey cancel each other out. We are then left with the basic approximation without any influence from the key.

**Subkey Extraction**

Differential cryptanalysis involves subkey extraction in exactly the same way as linear
cryptanalysis. When given the full-cipher differential approximation, a large number of plaintext/ciphertext pairs are partially-decrypted using all possible candidate keys (performing one round of decryption to obtain the input to the last round of S-boxes). The key difference between this step as compared to linear cryptanalysis is the need for a specific input differential—that is, differential cryptanalysis is a chosen plaintext attack rather than just a known plaintext attack.

Each decryption is examined for conformity to the differential approximation, and that key is assigned a bias. In differential cryptanalysis, contrary to linear cryptanalysis, a key's bias is the total percentage of pairs that conform to the approximation. As with linear cryptanalysis, the key with the largest absolute bias is assumed to be the correct key.

**Literature Review**

Although no tools currently exist to provide an interactive, visualized experience for linear and differential cryptanalysis, there do exist a number of other applications that provide visualization and animation for various algorithms, including cryptographic algorithms. Many of these tools are specifically intended to be educational tools, which makes them particularly relevant to the development of my own application. In this section I will denote a number of referenced papers and existing tools for interactive, visualized educational tools, and indicate the lessons learned from each publication.

**Anane**

This paper[2] was perhaps the most comprehensive with regards to design considerations, and had the most evaluation criteria which could be adopted by this project. Although the paper was primarily focused on teaching DES, the criteria it outlined were sufficiently abstracted as to be generally applicable. For the sake of organization, I categorize the following observations for Anane et al.'s design considerations according to the way they originally categorize them in the paper. I have attempted to generalize this paper's observations about their own algorithm animation, such that they may be better adapted to my particular implementation.

- **Context**
  - The tool should be self-contained, and supported by relevant information in-context within the tool itself, rather than depending on external reference.
  - With regards to relevant information, an attempt should be made to, to the greatest degree possible, keep notation within the realm of mathematical standards, avoiding any unexplained and unconventional notations.
  - The tool should be reliably deterministic and consistent, such that given a series of identical inputs it can be expected to receive the same output.
  - The tool should be variable in complexity, such that the user can alter their input to receive
more simplified or more complicated execution.

- The tool should be able to demonstrate the value of its concepts; for example, in the case of the DES tool outlined in the paper, the goal was to demonstrate the “intrinsic value” of DES and of cryptography in general through use of the tool.

- **Interaction**
  - Each visual object should have a well-defined purpose and focus within the tool.
  - The tool should support varying levels of granularity and abstraction regarding which components it visualizes.
  - The tool should minimize the amount of information the user is required to retain between stages of execution.
  - The tool should have as near as possible to a one-to-one mapping between a visual object and an abstract or conceptual object/step.

- **Usability**
  - Users should be informed of progress through the different stages of the tool and execution of processes.
  - The tool should be clearly navigable and usable.
  - The tool should implement a minimalist design, without any extraneous details that distract from its purpose.

**Fuzi**

The animation described in this paper[3], which focused on differential cryptanalysis of Simplified Data Encryption Standard (SDES), appears to be a much more generic animation tool than many of the other papers reviewed for this project report. Unfortunately, I am limited in my ability to thoroughly assess the tool, as I was unable to locate a copy of the tool to test firsthand and am limited to the paper as reference. By the descriptions given in the document itself, it appears to suffer from a lack of customization and user interactivity—because it appears that all variables for the process are in fact constant in the tool, the result ends up being more of a fancy slide-show than an interactive animation.

However, this approach and its justifications provide some insights into the production of a more robust and flexible interactive tool that may otherwise be overlooked when getting caught up in the process. Their solution provides a simple animation with background information and details, as well as in-tool theory explanations. Some important criteria can be discerned from their design, modeled after the following features:
• Definition of terms and rudimentary information on basic SDES algorithm components provided in-tool.

• Information specific to the cryptanalysis technique is explained, and a summary of more granular details given.

• Cryptanalysis animation can be viewed at different levels of abstraction or detail.

• An appendix is provided which gives information on what variables the tool used, such as the values for its substitution tables or subkey values.

Although the goal of this project was obviously to create something somewhat more substantial than this particular algorithm animation, it provides a good foundational framework for how to keep the information relevant to the project self-contained and bearing in mind to minimize a user's reliance on external reference information.

Hu

This paper[5] examines the pedagogical viability of visualization for cryptographic algorithms. In its own review of existing literature it comes to an interesting conclusion: gamification of learning in the field of computer science is effective, but algorithm visualizations in particular are largely failures, especially with regards to usability and stability—this emphasizes the importance of a well-conceived design with clear goals and evaluation criteria.

Some of the characteristics particular to this framework design are:

• Compatibility with most symmetric and asymmetric ciphers.

• Explanation of granular components of algorithms, such as S-boxes or key generation methods.

• Interactivity, with flexibility for variation on key size, input data, etc.

• The ability to toggle different visualization features on and off.

• The framework attempts to support both preexisting and novel ciphers:
  ◦ Preexisting ciphers have a number of input parameters defined and construct a visualization from there.
  ◦ Novel ciphers are constructed by the user dynamically through selection and creation of different components, such as S-boxes or permutations.

Many of these principles and characteristics are easily generalized to an LDC tool, particularly as some emphasis was put on the example of DES as a supported animation—visualization of a SPN and its subsequent cryptanalysis is logically compatible with this principle.
Picek

With a command-line and a C-library version, this sort of interactive tool[9] doesn't conform to the idea of a visualization tool. At a very basic level, the following features of this tool are of interest:

- Arbitrary input/output sizes are supported where mathematically possible.
- The authors of the tool attempted to support as many features of cryptanalysis as possible, and provide as much detail and calculate as many properties about the S-boxes as possible.
- The program supports calculation of as many or as few properties as requested.

Program input is read from files, and output is written to files.

Tao

This paper[9] examines the use of a specific tool which visualizes the Data Encryption Standard (DES) cipher, for the purpose of instruction in a mathematics-oriented course, “MA3203 Introduction to Cryptography,” at the Michigan Technological University. As a part of its usage, a survey was conducted amongst students in the class, with the following questions (quoted verbatim):

1. “DESvisual helped me see the inner workings of the DES cipher.”
2. “I understand the DES cipher more after I was able to use DESvisual.”
3. “With DESvisual I was able to identify the parts of the algorithm that I did not understand.”
4. “The demo mode was helpful for my self-study.”
5. “The practice mode helped me understand the DES operations.”
6. “DESvisual enhanced the course.”

Although these questions pertain specifically to the DESvisual application, they can be abstracted and generalized to pertain to a general cryptographic learning tool, and further specifically to an interactive LDC tool.

Aside from the survey questions, which provide decent general evaluation criteria, there were a number of user-provided comments which also provide perspective on the implementation in the paper and can shed lights on its shortcomings, which is useful in the development of another cryptographic learning tool. Particularly, the comments denoted as “major problems” were (again, quoted verbatim):

1. “Support 12-bit input and full DES cipher.”
2. “Should use symbols and notations exactly the same as the simplified DES algorithm in the textbook.”
3. “The designs are overly complex.”

This echoes several sentiments from the Anane paper, notably those principles of simplification,
DiffCryptDemo\textsuperscript{[7]} is a tool implemented in Excel with a supporting macro. Because tabular data is useful for defining S-boxes and permutations, this aspect works well. However, it is light on explanation of theory—although it contains explanations of differential cryptanalysis concepts which assume little prior knowledge of the attack, enough is left unexplained that meaningful use of the tool would still require an external reference for a student. Even the explanation of the tool itself on its “How it works” tab has its faults, and even when supplemented by the tool’s README file, it requires playing with the tool a bit to get the hang of it. As this tool is explicitly designed for use in a classroom setting, this is a mixed bag: on one hand, it is intended to be used in conjunction with a formal lecture, which would likely explain use of the tool, and the fact that it takes a bit of getting used to may encourage continued playing by students; on the other hand, the lack of instruction for what is not necessarily an intuitive interface may confuse students and discourage continued use.
Another piece of information to note is the sheer volume of information in the “Let's Try It” worksheet. There are hundreds of rows of plaintext/ciphertext pairs generated. This appears with little information accompanying it in the instructions, which may cause students to be confused by what they represent if they are beginners to differential cryptanalysis. There is also a verbiage issue in the cell which announces the found key, where quite a bit of information is given without explanation. Since this is implemented in Excel, it may have been wiser for that information to have been made tabular, so that row and column labels could better explain the result of the tool. However, despite its shortcomings, DiffCryptDemo is a decent instructive tool for such a simple and limited implementation.

**Evaluation Criteria**

After performing the literature review and discerning the successes and failures of prior algorithm animations and pedagogical cryptography applications, I strove to create a set of design evaluation criteria by which to design my tool, as to achieve optimum effectiveness. These principles also represent details which can be concretely evaluated against the finished product in order to discern the finished product's faithfulness to the foundational ideas upon which it was designed and built.

This section outlines the evaluation criteria for the UI design of the project as determined by review of the literature. As cryptographic principles are some of the most difficult to learn and most likely to meet resistance from students[6], it becomes a great concern to create a usable and helpful
visual learning tool that is not itself a burden to learn or use.

The following criteria are aggregated from the literature review of the Analysis section by taking those aspects which are most applicable to the project at hand and which would provide the greatest benefit for their effort. The criteria are organized into two categories: requirements and considerations.

**Requirements**

In this project, “requirement” is used to refer to an evaluation criterion considered necessary for completion of the project. They therefore must be complete on every applicable component, screen, or process. The requirements for this project are:

1. Each screen on which a user defines a cryptographic component must contain a button which, when pressed, will provide a conceptual explanation of that component.

2. For the purpose of clarity, any component using mathematical notation will remain consistent with the notation used in Heys's tutorial\[4\]. In any case where such notation does not appear in the tutorial and is not rudimentary, notation will be selected on a basis of best-effort to identify convention. In the event more than one or two such situation present themselves, there shall be an in-app reference provided.

3. Each dialog shall serve a single purpose; for example, a dialog to define a permutation shall not perform any operation exclusive to another component such as S-boxes.

4. Where possible and nontrivial, the application should have the capability to use a defined component in isolation.

5. The application should support arbitrary parameters where mathematically possible and within reason (for example, sizes which result in unnavigable UI or which will cause memory/performance issues might not be supported).

6. Cryptanalytical properties of ciphers and components shall be exportable to an external, human-readable file format, such as plain-text or HTML/CSS.

**Considerations**

In this project, “consideration” is used to refer to an evaluation criterion which is either desired but not strictly necessary for successful completion of the project, or else is not easily quantifiable, merely representing a concept to adhere to. Considerations for this project include:

1. UI design should strive to be minimalist and utilitarian rather than flashy; where a design choice is more aesthetic than functional, the decision should reflect what is most likely to result in a legible and navigable application.

2. The application should provide enough internal reference material such that user dependence on external reference such as the Heys tutorial is minimized.
3. Operation of the application should strive to be transparent about processing, such that it avoids
the impression of clunkiness or hanging where long operations are performed, such as bulk
cryptions/decryptions.

4. The primary goal of the application being to assist in cryptographic education, few assumptions
should be made about the existing knowledge of the user. While it should be expected that the
user knows such rudimentary terms as “cipher” or be familiar with non-cryptography-specific
concepts such as bit-operations (XOR, AND, etc.), more complex or cryptanalysis-specific
terminology should be defined. Conversely, an experienced user should not feel overburdened
by mandatory explanatory dialogs.

Project Design and Implementation

Overview

The project is implemented in Java using a Swing interface. It is built using Gradle, including
the 'application' plugin in order to be runnable outside of an IDE; it is run using the “gradlew ui:run”
command.

External Libraries

External tools and libraries used at this time include:

- Gradle – For easier build management and consistency between devices.
- JUnit – For effective unit testing and early detection of introduced bugs
- JaCoCo – For code-coverage reports of unit testing and assurance that all necessary test-cases
  are covered.
- Apache Commons Lang 3 – For various miscellaneous utilities, such as ArrayUtils.reverse()
  and the like.
- XStream – Used to serialize user-defined SPNs in a human-readable and human-editable way.

Project Structure

The Gradle project structure is split into two subprojects:

- CORE – This is the computational backbone of the overarching project. This project contains
  the linear/differential cryptanalysis (LDC) and substitution-permutation network (SPN)
  implementations, as well as persistence annotations for XStream.
- UI – This project contains the entire Swing structure connecting the user to the back-end code
  and visualizing the process.
SPN Construction Structure
My implementation for the substitution-permutation network involves the following classes:

- **SBox** – Backed by an integer array; maps input values to output values and maintains a linear approximation table (LAT) and difference-distribution table (DDT) for each object.
- **Permutation** – Backed by an integer array; maps input positions to output positions using BitSets
- **Key** – Simply holds a value and allows XOR operations with inputs
- **Round** – Contains a Key (representing a subkey), a set of SBoxes, and a Permutation, and delegates operations on inputs to each of those objects in that order.
- **SPNetwork** – Contains a set of Rounds applied in order.

This design structure allows each Round to have a distinct set of operations if desired. For example, each Round could have a separate permutation from the others, or even SBoxes of different sizes (for example, Round 1 may have 4 4-bit SBoxes, while Round 2 may have 2 8-bit SBoxes). Due to the need to ease test cases during limited development time, however, the UI currently enforces uniform S-boxes throughout the SPN.

LDC Structure
The implementation for the LDC structure involves the following:

- **AbstractApproximation** – an abstract class, extended by LinearApproximation and DifferentialApproximation, representing a full-cipher approximation including plaintext and semifinal-round masks. Its subclasses define methods to determine whether a given pair conforms to the approximation.
- **AbstractKeyBiasExtractor** – an abstract class, extended by LinearKeyBiasExtractor and DifferentialKeyBiasExtractor, which calculates a bias table and attempts to find a target partial subkey given a collection of plaintext/ciphertext pairs and a corresponding Approximation.
- **BiasExtractorProgressCallback** – an interface used to report back completion of each pair-check for each potential target partial subkey depending on its caller. For example, unit tests report progress to the standard output stream while the UI reports progress to a progress bar.

UI Structure
In this section I will outline the functionality and design of the Swing UI. The details of the tool will be explained and the different relevant dialogs shown.

Master Properties Dialog
Upon opening the tool, a user is presented with the option to either load a persisted SPN for cryptanalysis or to create a new SPN to cryptanalyze. If “Load” is selected, a JFileChooser dialog is used to select an XML file (exported previously from this tool using the XStream library, or written manually in the appropriate format). If a new SPN is selected, a user is prompted for the following parameters:

- **Block size, in bits**: This is the “width” of each round in the cipher, and how many bits at once constitute an encryptable block.
- **Num. Rounds**: The number of iterations for a given block.
- **S-box size**: The size, in bits, that each S-box accommodates. This number determines the number of S-boxes per round, and must evenly divide the block size, or else the following error will occur:

![ERROR Dialog](image)

**ERROR**

Block size must be divisible by S-box size.

OK

---

**S-box Definition Dialog**

![Edit S-box Dialog](image)

This dialog is intended for use as a means for a user to define their own component for the S-box part of a SPN. A simple dialog, it contains the following features:

- A number of text areas equal to the number of unique inputs and outputs for the S-box, each...
with corresponding labels in hexadecimal.

- Labels denoting the input versus output parameters.
- An OK button to accept and validate the changes, and a Cancel button to discard changes and leave the SPN as it was before the dialog was opened.
- A Help button which will produce a dialog intended to explain the purpose of S-boxes to the user and provide insight into the meaning of the input fields:
**Permutation Definition Dialog**

This dialog is intended for use as a means for a user to define their own component for the permutation part of an SPN. It contains the following features:

- A number of text areas equal to the number of bits in the block size is produced on the dialog, each with corresponding labels in hexadecimal.

- The user may map their permutation using the top row of labels as “in” and the text areas as “out.”

- There is a visualization above which vaguely resembles the visualizations in the Heys tutorial, with two straight rows of bits connected by lines. In this visualization each bit is represented by a “pin” shape, with a circle and a line protruding from it. The points of these pins connect with straight lines to form the visualization.

- An OK button to accept and validate the changes, and a Cancel button to discard changes and leave the SPN as it was before the dialog was opened.

- A Help button which will produce a dialog intended to explain the purpose of Permutations to the user and provide insight into the meaning of the input fields:
Cipher

Selection Dialog

This dialog provides the user the ability to define a new SPN manually or to load a preexisting SPN that has been serialized. It is a simple “New vs. Load” dialog. The buttons work as follows:

- **New** – This button produces the Master Properties Dialog, and from the properties provided to the tool a new SPN is created wherein each component is a non-operative (no-op) component. The visualization will be blank initially, as no-op components do not render.

- **Load** – This button loads a file chooser which allows the user to select an XML file, either previously persisted with XStream or manually written to the conforming format. The SPN defined in the XML will then be loaded into the tool and the rest of the operation will proceed as normal. Persisted SPNs do not retain cached values for named components (discussed on the
next section), although any existing components on the SPN could be immediately added to the new cache.

**SPN Definition Dialog**

This dialog allows a user to define the full beginning-to-end SPN cipher. It contains the following features:

- All components of the network are laid out in a tree format, representing the sequential components through which a bit string passes.
- Right-clicking on an entry for a component and selecting “Edit” will pull up that component's definition dialog.
- In the right-click menu, one may reference a previously-defined component in a new part of the structure, to prevent having to define an identical component multiple times. This is done through “Store Copy” and “Restore Named Copy,” which store references to the given component in a cache.
- The File menu contains a Save option which allows a user to persist their SPN as XML and load at a later date. This opens the opportunity for the inclusion of sample SPNs with known properties for use during demonstrations.
SPN Visualization Frame

This frame exists in conjunction with the SPN Definition Dialog, where the tree structure is visualized on a frame as a chart similar to the Heys tutorial depiction of a network (a representation also used in the DiffCryptDemo tool[7]). The intent of this dialog is to provide a more intuitive and informative visualization of the information defined in the SPN Definition Dialog. I felt it was necessary to separate the visualization from the definition in this case, as clicking through the graph to define components could be confusing. As a major concern with learning tools is their usability[4], it's also necessary to minimize potential that components may behave unpredictably, an issue that such a complicated interface as editing the SPN directly on the graph would invite.

The Visualization Frame also serves the vital role of demonstrating what is going on during the process of forming full-cipher approximations for LDC. As opposed to the Heys tutorial, which attempts to link a textual explanation of a component with a static image, this tool allows a user to explicitly define their approximation and watch the path of bits through the cipher appear instantaneously. This feature is demonstrated in the following section.
**Cryptanalysis Dialogs**

This set of dialogs forms the workflow for performing LDC on a defined cipher. The process for Linear Cryptanalysis is outlined in this section, although the process for Differential Cryptanalysis is nearly identical.

Upon choosing a Linear Cryptanalysis attack, a user is presented with a dialog outlining a grid of S-boxes which, when selected, will display a Linear Approximation Table (or Difference Distribution Table, in the case of Differential Cryptanalysis). Approximations of zero are not selectable, nor are selections which do not match the input mask from the prior round (selection of which would break the approximation). Upon selection of an appropriate approximation and selecting Accept, it will update the S-box table to disable any S-boxes which do not receive input from the prior round's mask (with the exception of the first round, which are always selectable):
In the example above, I show that S-box (1, 1) will receive all output from S-box (0, 1) and is therefore the only one enabled on that round. Similarly, S-box (1, 1) has output bits which enter both S-box (2, 1) and (2, 3), which enables both of those boxes on that round. When a full-cipher attack is defined, the “OK” button will attempt to determine the target partial subkey as defined in the construction of the approximation:

As with the component definition dialogs, each dialog during the LDC process contains a help button, which will produce a window denoting the purpose of the dialog and of the task which that dialog represents, in an attempt to help less-familiar users (such as introductory cryptography students) understand the process as they go through it.
Project Evaluation

Evaluation of UI Design Criteria

In this section I will analyze the final UI components created for this project against the requirements listed in the Evaluation Criteria section. Each number listed below will refer to the number of the original requirement.

New/Load and Master Properties Dialogs
The following relevant requirements are met by this segment of the tool:

1. Although this screen does not define a cryptographic component, it provides information on the information needed to construct it where appropriate.

3. Each dialog in this part of the program serves a single purpose.

5. The application supports arbitrary parameters up to a 64-bit block size, although not where mathematically impossible. Extremely large sizes are largely untested, both because they are not useful and because the UI is limited by screen resolution for absurdly large visualizations and approximation tables.

Areas of improvement include alignment of labels and fields to provide a uniform feel. However, this was not considered essential functionality for minimum viable product, and will be discussed in Future Work.

SPN Definition Dialog
The following relevant requirements are met by this segment of the tool:

1. Although this screen does not itself contain a help button, it also does not define a specific component. Each subsequent definition dialog for individual components does contain a help button.

3. This dialog only attempts to do one thing, which is to manage the SPN defined or loaded by the user. This includes multiple use-cases, such as reaching dialogs to define sub-components, persist the cipher, or perform cryptanalysis.

6. Properties of the cipher can be exported to a readable format in that it is serializable into XML through XStream.

SPN Visualization Frame
The following relevant requirements are met by this segment of the tool:

2. Although not strictly notation, the drawing of SPN components and the method of highlighting
has been directly based on the Heys tutorial.

3. The frame serves a single purpose in displaying the current status of the user-defined SPN. “Current status” in this case encompasses the approximations of the LDC process, as that is the state of the SPN from the perspective of the user at that point in the workflow.

**Component Definition Dialogs**

The following relevant requirements are met by this segment of the tool:

1. Each definition dialog contains a button which, when pressed, can provide a conceptual explanation of that component.

2. Mathematical notations have remained consistent with the notation used in Heys’s tutorial and/or common convention.

3. Each dialog serves a single purpose.

   These dialogs can be improved with correction of spacing issues, although largely they are satisfactory the way they are. A design choice was made with these dialogs regarding the use of components in isolation—because the dialogs themselves so clearly represent the operations performed on input, inclusion of a “test it out” functionality on the dialogs would be cluttered and could confuse an end-user who does not make use of the help button.

**Cryptanalysis Dialogs**

The following relevant requirements are met by this segment of the tool:

1. Help buttons are present on relevant dialogs.

2. Demarcation of individual S-boxes is similar to the method used in Heys. We chose to put positions in parentheses rather than subscript for simplicity and readability.

3. Each dialog relevant to the cryptanalysis process has a clearly defined purpose with no extraneous usage. Any significant step in the process is given its own dialog.

4. Cryptanalysis allows arbitrary approximations for both linear and differential attacks where such approximations are mathematically valid and nontrivial (i.e., deviate from chance).

   Although minimalism was encouraged by the design requirements and considerations, some of the main approximation dialog (where S-boxes are selected) is perhaps detrimentally minimalist. While there exists a help dialog as with the other windows, it could do with some relabeling or alternative structure to better provide an intuitive relationship between the buttons on the screen and the components in the SPN.

**Retrieval of Target Partial Subkeys**

Subkeys are successfully retrieved on a consistent basis for both linear and differential
cryptanalysis. Although they are not perfect, the process itself is not perfect for most ciphers, and depending on S-box strength the process may not be effective at all. On sample ciphers known to be susceptible to LDC attacks, the tool is able to guess the full target partial subkeys the vast majority of the time, and when it is incorrect it is typically only off by one bit. An example is shown below of differential cryptanalysis output from the tools Junit tests:

<table>
<thead>
<tr>
<th>Key 256/256 Pair 4991/5000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key 256/256 Pair 4992/5000</td>
</tr>
<tr>
<td>Key 256/256 Pair 4993/5000</td>
</tr>
<tr>
<td>Key 256/256 Pair 4994/5000</td>
</tr>
<tr>
<td>Key 256/256 Pair 4995/5000</td>
</tr>
<tr>
<td>Key 256/256 Pair 4996/5000</td>
</tr>
<tr>
<td>Key 256/256 Pair 4997/5000</td>
</tr>
<tr>
<td>Key 256/256 Pair 4998/5000</td>
</tr>
<tr>
<td>Key 256/256 Pair 4999/5000</td>
</tr>
<tr>
<td>Key 256/256 Pair 5000/5000</td>
</tr>
</tbody>
</table>

Found target partial subkey [0000]

Linear cryptanalysis, and by extension this tool, will fail on a cipher with exclusively non-operative (identity) S-boxes. However, such ciphers are strictly linear by definition, and are therefore uninteresting and trivially broken without the use of advanced cryptanalysis techniques. The mathematical reason that such ciphers will fail to be cryptanalyzed in this manner is due to the way the bias is calculated in linear cryptanalysis. Because the bias is taken as the absolute value of the difference from chance:

$$|bias| = \frac{|matchCount - \left(\frac{numPairs}{2}\right)|}{numPairs}$$

The resulting bias from one key value will be $|0.5|$ and all others $|-0.5|$. These keys are considered to be equally likely, and so the result calculated is dependent only on how the tool determines which to choose in the event of a tie. In the case of Maledict, it simply chooses the one it first comes across, and since it tests all keys in order, it will always return an all-zero key for a cipher consisting entirely of no-op S-boxes.

**Example Report**

The report for a cryptanalysis coalesces vital details both about the cipher itself and the results of the analysis. Notably it provides:

- An outline of the structure of the SPN.
- The “path” selected through the SPN for the cryptanalysis attack.
- The input and last round mask (in the case of linear cryptanalysis) or differential (in the case of
differential cryptanalysis).

- The linear approximation table (for linear) or difference distribution table (for differential) for the S-boxes.
- A list of the top 10 subkey biases and their corresponding subkeys, indicating the largest bias as the assumed target partial subkey.

A partial example of a cryptanalysis report is shown below:

**Cryptanalysis Report**

**Cipher Approximation Masks**

<table>
<thead>
<tr>
<th>Mask</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plaintext</td>
<td>101100000000</td>
</tr>
<tr>
<td>Last-Round</td>
<td>10100000101</td>
</tr>
</tbody>
</table>

**Top Key Matches**

<table>
<thead>
<tr>
<th>Top Biases</th>
<th>Key</th>
<th>Bias</th>
</tr>
</thead>
<tbody>
<tr>
<td>0204</td>
<td>0.028300</td>
<td></td>
</tr>
<tr>
<td>0206</td>
<td>0.023700</td>
<td></td>
</tr>
<tr>
<td>0704</td>
<td>0.021900</td>
<td></td>
</tr>
<tr>
<td>0202</td>
<td>0.020700</td>
<td></td>
</tr>
<tr>
<td>0304</td>
<td>0.020300</td>
<td></td>
</tr>
<tr>
<td>0006</td>
<td>0.019800</td>
<td></td>
</tr>
<tr>
<td>0102</td>
<td>0.018900</td>
<td></td>
</tr>
<tr>
<td>020A</td>
<td>0.018700</td>
<td></td>
</tr>
<tr>
<td>0302</td>
<td>0.018100</td>
<td></td>
</tr>
<tr>
<td>0508</td>
<td>0.017500</td>
<td></td>
</tr>
</tbody>
</table>

**Cipher Visualization**
Development Roadblocks

The latter portion of the project development accumulated a number of issues which impacted the timeliness of deliverables and subsequently the amount of extra functionality and material that I was able to include in the finished product.

SPN Drawing Issues

The original design for the SPN Visualization Frame involved total reuse of code from the Permutation Definition Dialog's visualization. To that end, an attempt was made to separate each SPN component into a separate geometric object. These would then be displayed and arranged on the screen. The results were ultimately not satisfactory:

This failure to properly render the visualization was due to many underlying issues in the design:

- **Scaling** – The fact that each component was rendered as a separate object on the panel meant that some mechanism needed to exist to maintain a consistent scale between user-space coordinates and device-space coordinates, not for the entire frame, but for each individual component. This could be extremely tricky, particularly when Swing's layout managers do not always make it clear what exactly is going on behind-the-scenes.

- **Alignment** – In the same vein, components maintaining their own sizing information meant that even if the layout manager could be coaxed to place everything precisely where it was needed, the drawing scale could be off just so slightly that all but one or two lines would be lined up, and those would be off by a pixel. In other cases, there may be a one-pixel variance between touching lines.

- **Spacing** – Distinct from the issue of alignment between adjacent components and lines, it was difficult to get the components to be exactly touching, with no blank space between them. This issue was ultimately, I think, linked to scaling issues, where some components were reporting themselves as taller
than they were drawing themselves. Since this design was scrapped, however, that cause is not certain.

- **Performance** – Because each component was calculating everything for itself, separately, updating and resizing the SPN Visualization Frame could create a small but noticeable lag, going as far as to briefly lock up the UI while it calculated what it needed to do.

  Ultimately, the solution was to scrap the componentized design of the SPN visualization, and unite it under one component. This meant duplication of code between the Permutation Definition Dialog and the SPN Visualization Frame, but to far better effect. With the implementation complete, the process to properly deduplicate that code in a way that makes sense and works for both components is simple, and could ultimately be a technical debt owed to future development.

**Data Type Conversions and Endianness**

When moving data between the UI code and the underlying framework for cryptanalysis, a number of data conversions have to take place in order for the code to be both performant and understandable. Initial code, which relied only on arrays of the primitive byte type, behaved predictably. However, I later introduced the use of long primitives for use in LDC components, partially for simplicity and partially to make it easier and more reliable to generate at the UI level. As 64 bits was a more-than-reasonable restriction on block size for this tool, I initially had no qualms about introducing this functionality and refactoring some of my code to make use of longs rather than bytes.

However, my use of BitSet in a number of places, as well as some necessary conversions between the longs and the bytes, created an at-times-confusing web wherein often something would come out wrong or in the incorrect order. The convenience of longs for bitwise operations necessitated their use, but the effectiveness of BitSet for bit-permutations was also too good to pass up. Ultimately, it came down to borrowing a method from Stack Overflow (credited in in-code documentation) to convert BitSet to a byte array in a manner more consistent with my expectations than the built in toByteArray() method. This, paired with the reverse() method in the Apache Commons's ArrayUtils class, ultimately solved the bit-ordering issue upon conversion between data types.

**Conclusion**

**Future Work**

Although I consider Maledict to have been a successful first step toward creating an effective interactive educational tool for LDC, it is not perfect. A number of significant improvements could be made to the tool upon continued development which would better orient the tool to both use in a classroom environment and as a practical tool for real-world LDC applications.
**Unknown Key Mode**

Perhaps the most major functional drawback of Maledict as a tool is its lack of an “Unknown Key Mode.” That is, plaintext/ciphertext pairs can only be generated within the tool using the key entered when defining of the SPN. There is currently no mechanism to pre-generate these pairs, for example from another program, and then import the pairs to Maledict. Because linear and differential cryptanalysis obviously do not require knowledge of the key (it is only needed to generate the pairs used for the attack), it would be a helpful expansion of functionality to be able to define the structure of a SPN without defining a key, and then provide external plaintext/ciphertext pairs (which have been encrypted with the actual cipher, using the real key).

Ideally, Unknown Key Mode would present as a flag on the creation or editing of a SPN, in which all Key components would be disabled, while S-boxes and permutations would still be defined. Upon performing Linear or Differential Cryptanalysis upon the defined cipher, the tool would need to deserialize and parse provided plaintext/ciphertext pairs and test the approximation against those, with the rest of the process continuing normally.

Although in practice the tool is blind to the Key when cryptanalyzing a cipher, Unknown Key Mode presents a more impressive demonstration to users who have not read its code and would be able to better trust the results when they know they have never provided the key for a given cipher to the tool. This would also expand its use as a legitimate cryptanalysis tool, as in many practical cases the user may have a number of plaintext/ciphertext pairs available, and know the structure of the cipher, but not know the key used to encrypt them.

**UI Improvements**

A number of improvements stand to be made on the UI as well. Although the vast majority of dialogs in the tool support resizing, the default sizes may be strange, with components being cramped together. This could be improved by testing the tool more comprehensively across platforms and screen resolutions.

A more concrete example of what stands to be improved on the UI is the separation of the SPN Definition Dialog and the SPN Visualization Frame. Upon initial design, the visualization was constructed in a much more fragile manner. It made sense, at the time, to separate the visualization from the main dialog in order to minimize interference from other components, or from resizing by the user. Keeping them separate enabled resizing of the SPN Definition Dialog in the first place, as the original design of the SPN Visualization Frame did not commit to being resizeable as a requirement. After the change in how the visualization is rendered on the screen, there is no longer any purpose for these to be on separate dialogs, and their separation hinders user experience by allowing the SPN Visualization Frame to fall behind other windows in the operating system.

**Class Trials**

As a primary intent in the development of Maledict was to create a tool usable in a classroom
User trials would be the ultimate test of Maledict's usability as an application in general as well as its pedagogical usefulness. As the developer of Maledict I am quite familiar with the concepts of LDC, and try as I may to assume minimal knowledge of the user in my creation of the tool, it is inevitable that some thing or another has been overlooked, under-explained, or otherwise made confusing. And despite the fact that the tool was designed based on concepts gleaned from prior cryptographic educational applications and algorithm animations, my consideration of those concepts is not a guarantee of empirical success. Similarly, where I have diverged from the failures of the existing literature, that divergence is not a guarantee of improvement. User trials among the target audience are a better way to assess whether the tool is ultimately a success.

Closing Remarks

This paper has demonstrated and explained a tool for the learning of linear and differential cryptanalysis techniques on substitution-permutation networks. The pedagogical value of such a tool in concept, a tool which breaks out the steps of the process interactively with accompanying help material, and which visualizes the cryptanalysis process along with the user's steps, is clear.

Linear and differential cryptanalysis are essential for modern cryptographic education, as any new SPN ciphers will inevitably be tested for resistance against these attacks, and knowing these attacks is important in understanding the design of modern ciphers such as AES, which were designed for such resistance. And in a field of study as difficult as cryptography, opportunities to incorporate both visual and tactile learning experiences are to be celebrated as supplementary material to written tutorials such as Heys\textsuperscript{4}. With regards to linear and differential cryptanalysis, which have precious few tools designed as learning aides for them, Maledict clearly expands the field of cryptographic instructional tools available to educators.

Although there has not yet been an empirical evaluation of its efficacy in a classroom environment, a solid foundation has been established, upon which continued development and feedback could someday mold it into an essential tool for introductory-level cryptography education.

References


