Maximizing Privacy with Minimal Secrecy

Mike Freyberger

Electrical Engineering
Princeton University
Junior Independent Work
Adviser: Prof. Paul Cuff
Second Reader: Prof. Prateek Mittal

Submitted in partial fulfillment
of the requirements for the degree of
Bachelor of Science in Engineering
Department of Electrical Engineering
Princeton University

June 2015

I hereby declare that this Independent Work report represents my own work in accordance with University regulations.

/s Mike Freyberger

Mike Freyberger

Abstract

Maximizing Privacy with Minimal Secrecy
Mike Freyberger

In order to communicate in a secure manner, there must be a method of making the communication system confidential. Confidentiality is typically achieved via a method of encryption. There are methods of using symmetric and asymmetric keys in order to make the system confidential. However, using asymmetric encryption is very slow, and maintaining symmetric keys can often be difficult. Symmetric keys are typically 128 bits, and must be refreshed frequently. What if the amount of secrecy between the two parties is severely limited? In this case, an encoder must be constructed such that the eavesdropper's probability of error, also known as the eavesdropper's Hamming distortion, is maximized. This paper discusses how to construct an encoder that maximizes the eavesdropper's probability of error when there is only one secret key bit. This method is then extended to study the case in which there are $lg \mid 3 \mid$ bits of secrecy. Lastly, this paper studies how introducing a stochastic encoder can increase the eavesdropper's probability of error, and how variable length encodings influence the proceeding results.

Acknowledgments

I would like to thank my adviser Prof. Cuff for all of his support and help throughout the semester. It has been awesome working on this project with him!

I would also like to thank Sanket Satpathy, a PHD student in Prof. Cuff's group. Sanket was always very helpful and took his time to explain his original work in the area, and helped me develop my own contributions to the project.

I would also like to thank Amy Cutchin for helping me proofread my final paper.

1 Motivation

Claude Shannon defines perfect secrecy as a cryptosystem which encrypts any plaintext message X, into cipher text Y with the following property: $\boldsymbol{P(X|Y) = P(X)}$. However, in order for a system to achieve perfect secrecy, the size of the key must be at least as big as the size of the cipher text: $\boldsymbol{|K| \geq |Y| \geq |X|}$. Therefore, this type of communication system has stringent requirements for the amount of shared secrecy.[1]

With a large amount of shared secrecy, encryption schemes typically use a simple one time pad. A one time pad XOR's the plaintext with the key in order to produce the cipher text, and XOR's the cipher text with the key in order to recover the plain text. This scheme works well when the amount of shared secrecy is large. But, what if the amount of shared secrecy is limited? For instance, what if the two communicating parties only share one bit of secrecy? It is currently unclear what encoding scheme would increase the probability of error at the eavesdropper. This paper seeks to give insight into how an encoding scheme can maximize the eavesdropper's probability of error when there is minimal secrecy.

2 Problem Setup

Typical communication systems, with a passive eavesdropper, look like Figure 1. $\boldsymbol{X}$ is the input source. $\boldsymbol{X}$ is a discrete random variable with a known probability distribution of $\mathrm{P}_{\boldsymbol{x}}$. A is the encoder, B is the decoder, and E is the eavesdropper. The encoding and decoding scheme in A and B are publicly known. $\boldsymbol{\hat{X}}$ is the value returned by the decoder. In an error free communication system $\boldsymbol{P(X = \hat{X}) = 1}$. Z is the value produced by the eavesdropper, and the goal is to produce an encoding and decoding scheme that maximizes $\boldsymbol{P(Z \neq X)}$. Lastly, M is the bit pattern that was used to encode the input source symbol. Notice we have N bits of communication. S is the secret bit pattern, which is limited to K bits. S is the only piece of information that is not made publicly known to the eavesdropper.

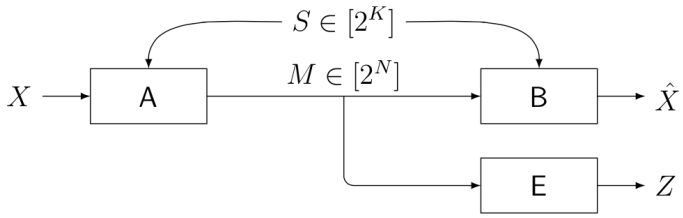

Figure 1: The problem set up.

2.1 Input Source

A discrete random variable, $\boldsymbol{X}$, is provided to the encoder A. The discrete random variable has a publicly known probability distribution $\mathrm{P}_{\boldsymbol{x}}$. Therefore both the receiver and the eavesdropper know the input distribution of the source. We assume the input source symbols are not correlated. Therefore, the following statement is true: $\boldsymbol{P(X_{k+1}|X_k) = P(X)}$. In other words, each symbol sent in this communication system is completely independent of all other symbols that are sent.

2.2 Encoder and Decoder

When designing the encoder it is very helpful to visualize the encoding scheme as a bipartite graph. The bottom nodes are the bit patterns for each encoding, and the upper nodes are the input symbols. Each input

[1] Most communication systems today do not achieve perfect secrecy; rather, they have a shared secret that is large enough to make decoding at the eavesdropper computationally prohibitive.

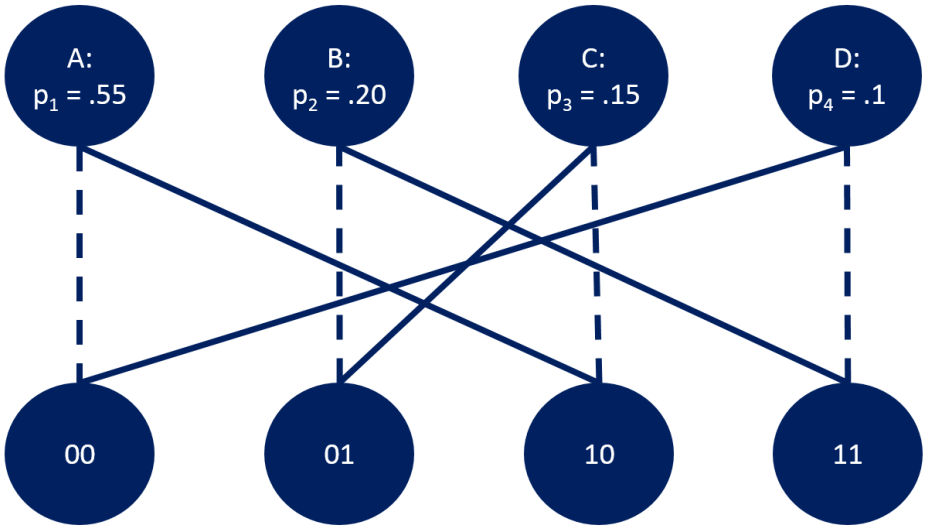

Figure 2: A bipartite graph representing an input source, and a possible encoding scheme. There is one bit of secrecy. Dashed lines correspond to $\boldsymbol{S}$, the secret key value, equaling 0, and solid lines correspond to $\boldsymbol{S}$ equaling 1.

symbol will connect to two bit patterns in the case that there is one bit of secrecy. In the case that there are two bits of secrecy, each bit pattern will connect to four input sources. An example of this bipartite graph can be found in Figure 2. In Figure 2 dashed lines correspond to the secret key value, $\boldsymbol{S}$, equaling 0, while solid lines correspond to $\boldsymbol{S}$ equaling 1. Therefore, if the input symbol is an A, the encoder will either send a 00 or 10, depending on the value of $\boldsymbol{S}$. Since $\boldsymbol{S}$ is a shared secret between the sender and the receiver, the receiver will be able to decode the symbol with 100% accuracy, while there will be a certain amount of distortion for the eavesdropper, because the eavesdropper does not know $\boldsymbol{S}$.

In this problem the input source symbols are encoded with zero-delay. A practical example of this set up includes sending sensor measurements across a channel. Every sensor measurement will be encoded immediately, and then sent over the channel. This system is considered to be a zero-delay encoding scheme because it does not wait to send a batch of symbols together.

2.3 Eavesdropper Scheme

The eavesdropper will use a scheme that attempts to decrease the probability of distortion for each message that is sent. Because of the assumption that all symbols are uncorrelated, the eavesdropper will make his decision based on the current bit pattern alone. In order to decrease his probability of distortion, he will always assume that the bit pattern that he reads on the communication channel is the most probable input source that could map to that encoding. The eavesdropper's guessing pattern is demonstrated by the red arrows in Figure 3. Figure 3 demonstrates that if the eavesdropper reads a 00 or a 10, he will guess that the original message was A, because that is the most probable input source symbol connected to both 00 and 10. Given this type of guessing scheme, the eavesdropper will have a certain amount of probability of distortion. For instance, in the case provided in Figure 3, the eavesdropper will never guess the message is a C or a D. Therefore, each of these input source values will cause distortion. The total amount of distortion at the eavesdropper is therefore $\boldsymbol{p_3 + p_4 = .15 + .1 = .25}$.

This encoding scheme is naturally worse than a scheme that meets the perfect secrecy requirements. If K, the number of secret key bits, equals 2, the messages in Figure 3 would be encoded with perfect secrecy. In this case, the eavesdropper would gain no information by seeing the cipher text, and would simply guess

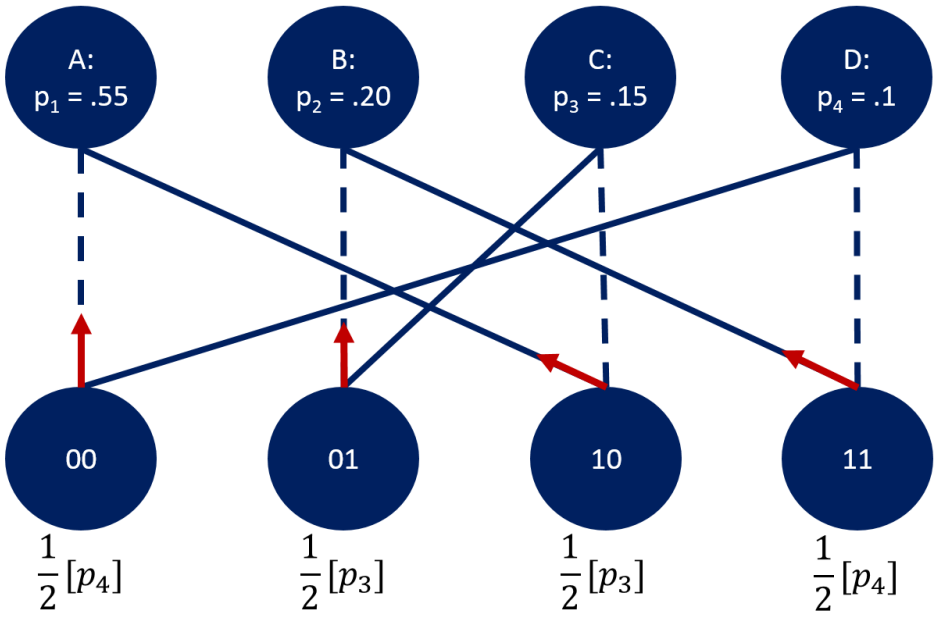

Figure 3: The eavesdropper's guessing pattern added to the original encoding scheme in red arrows. The eavesdropper will always traverse the graph as far left as possible (assuming the graph has the input source symbols sorted from largest to smallest). The distortion for each bit pattern is provided below each bit pattern.

the most probable input source symbol every time. Therefore, the eavesdropper's error in the perfect secrecy case is $\mathbf{1 - p_{max} = .45}$.

2.4 Upper Bound

In this type of privacy environment it is impossible to achieve 100% probability of distortion, because there are some fundamental limits governing the maximum amount of distortion in the system. First of all, the eavesdropper could just guess the most probable input source every time, which would lead to a distortion of $\mathbf{1 - p_{max}}$. Furthermore, if the eavesdropper guesses the secret key correctly, he will guess all of the values correctly. The probability of guessing the key correctly is $\mathbf{1 - 2^{-k}}$, where $\boldsymbol{k}$ is the number of secret key bits. Therefore, the upper bound for the amount of distortion that can introduced in this problem is:

$$\boldsymbol{min(1 - p_{max}, 1 - 2^{-k})} \tag{1}$$

If the encoding scheme achieves this upper bound, the scheme successfully achieves **maximal secrecy**. On the other hand, if the encoding scheme achieves the best possible distortion given the number of secret key bits and number of encoding values, the scheme achieves **optimal secrecy**. All schemes that achieve maximal secrecy are also optimal, but not all optimal schemes achieve maximal secrecy.

3 One-Bit Secrecy

3.1 Problem Overview

To begin, this is an encoding scheme that achieves optimal secrecy when there are only 2 secret key values, which corresponds to 1 bit of secrecy. [2] With 1 bit of secrecy, the problem is reduced to determining the

[2]This encoding scheme was developed by Sanket Satpathy, and the main points of the encoding scheme are presented here. For more details on this encoding scheme, including the proofs, contact him at satpathy@princeton.edu.

permutation that maximizes the eavesdroppers probability of error. In other words, imagine we are given the graph in Figure 4. This graph is incomplete because it only has the encoding scheme set up for when the secret key value, $\boldsymbol{S} = \mathbf{0}$, as that corresponds to the dashed lines. Solid lines must be added to this graph, representing the encoding scheme for when $\boldsymbol{S} = \mathbf{1}$. Also, we can start with this incomplete graph, rather than an empty graph, because any other permutation for these initial four lines essentially yields the same graph. Permuting these initial four lines is simply permuting the encoding bit patterns, which have no real significance to the problem. It doesn't matter if the bottom row is 00, 01, 10, 11 or 10, 11, 01, 01, etc.

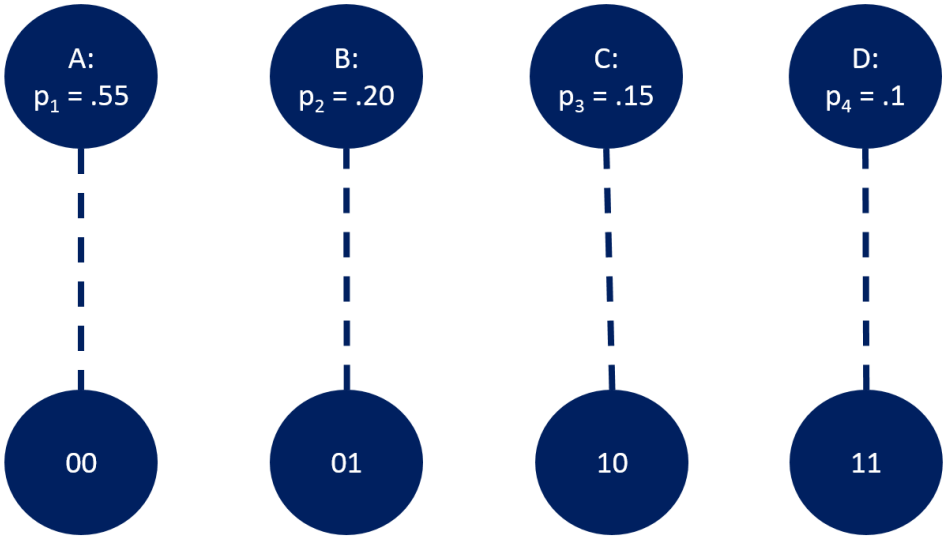

Figure 4: Incomplete graph for one bit of secrecy. The solid lines for $\boldsymbol{S} = \mathbf{1}$ must be defined.

3.2 Observations

In order to complete this graph we must determine the particular complete graph that achieves **optimal secrecy**. The search space for all possible graphs is $|\boldsymbol{X}|!$, which is super-exponential. In order to decrease the complexity of the problem, rather than search all possible permutations, only particular permutations are evaluated. In order to eliminate possible permutations, the first step is to realize the solution will be a set of cycles that are all disjoint, rather than one continuous graph. Furthermore, these cycles will always be in order and not overlap. For instance, if there are two cycles, one with $\boldsymbol{p_{max}}$ and $\boldsymbol{p_2}$, and the other with $\boldsymbol{p_3}$ and $\boldsymbol{p_4}$, this graph is in order and does not overlap. This type of graph is possibly optimal and is demonstrated in Figure 5. Whereas two cycles, one with $\boldsymbol{p_{max}}$ and $\boldsymbol{p_4}$, and the other with $\boldsymbol{p_2}$ and $\boldsymbol{p_3}$, will not be optimal because it overlaps. This type of graph is provided in Figure 6. Essentially, the cycles must not have any intersecting edges when the input source symbols are aligned in order. Furthermore, the size of the cycles will always be less than or equal to three input source nodes. An example of the incomplete graph in Figure 4 being completed with a 3-cycle is provided in Figure 7. The size can never exceed three input source symbols because any continuous graph with four input source symbols can be divided into two disjoint sub-graphs of two input source symbols in order to increase distortion. Furthermore, any cycle with more than 4 input source symbols can be split into some combination of 2-cycles and 3-cycles that increases the eavesdropper's probability of error. Therefore, the encoding scheme will consist of cycles with size two or three; however, it is possible to have a cycle of size 1 on the least probable input source symbol.

In summary, the optimal encoding scheme will consist of disjoint cycles, and the cycles will have the following properties.
1. Cycles are always in order.
2. Cycles do not overlap.

3. Cycles have length 2 or 3.

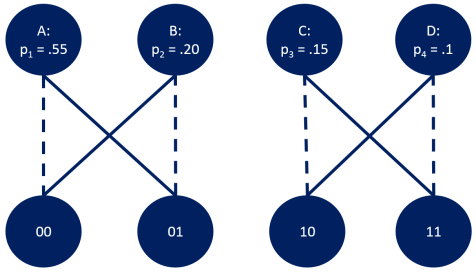

Figure 5: Possibly optimal set of cycles. 2 cycles of size 2.

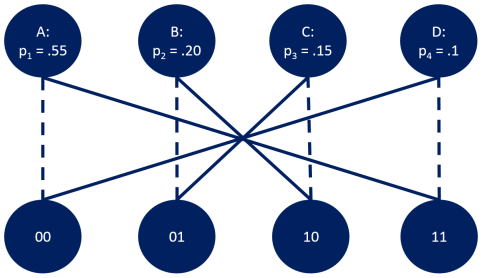

Figure 6: Non optimal set of cycles.

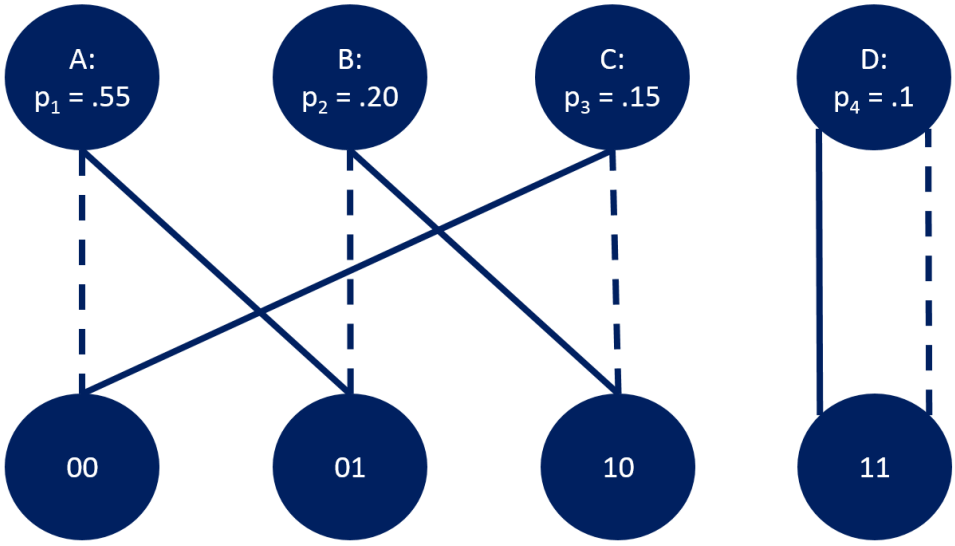

Figure 7: Possibly optimal set of cycles. 1 cycle of size 3 & 1 cycle of size 1.

3.3 Distortion Analysis

To begin the analysis of this encoding scheme, the distortion of cycles with size 1, 2, and 3 must be determined. For a cycle of size 1, both key values point to the same bit pattern, and there is no distortion introduced due to this cycle. The reason some permutations include cycles of size 1 is because sometimes it is best to optimize the $|\boldsymbol{X}| - \mathbf{1}$ nodes, and leave the last node to have zero distortion.

For a 2-cycle, we evaluate the left half of Figure 5, which is re-produced in Figure 8 with the eavesdropper guess. In this case, the eavesdropper never guesses the input source symbol with the smaller probability, so the distortion of a 2-cycle is $\boldsymbol{p_2}$, where $\boldsymbol{p_2}$ is the smaller probability with respect to the cycle.

For a 3-cycle, we evaluate the left half of Figure 7, which is reproduced with the eavesdropper's guess in Figure 9. In this case, the eavesdropper never guesses the least probable source symbol, and only guesses the second most probable source symbol for one of the key values. Therefore, the probability of error for a 3-cycle is $\frac{\boldsymbol{p_2}}{\mathbf{2}} + \boldsymbol{p_3}$, where $\boldsymbol{p_3}$ is the smallest source symbol in the cycle and $\boldsymbol{p_2}$ is the second most probable symbol in the cycle.

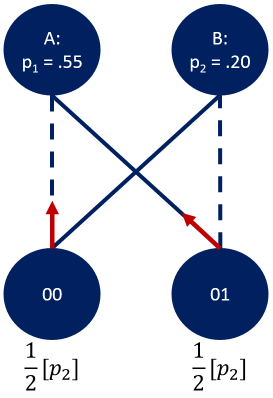

Figure 8: A 2-cycle, with the eavesdropper's guessing pattern and distortion.

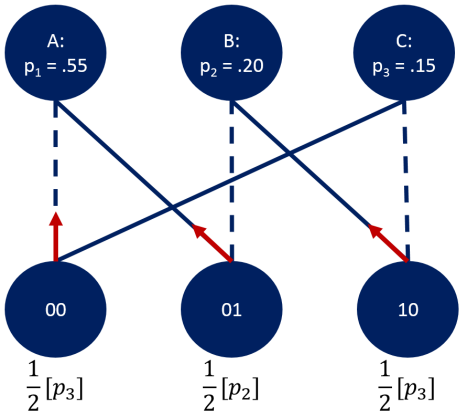

Figure 9: A 3-cycle, with the eavesdropper's guessing pattern and distortion.

3.4 Example

For the example that has been reproduced throughout this section, this section evaluates the possibly optimal graphs, and determines the optimal encoding scheme. For this case, since there are alone 4 input source symbols, there are only two possible optimal graphs. The encoding scheme will either be 2 cycles of size 2 (provided in Figure 5, or 1 cycle of size 3 and, 1 cycle of size 1 (provided in Figure 7).

Case 1: 2 Cycles of 2 leads to $\boldsymbol{p_2 + p_4 = .3}$.
Case 2: 1 Cycles of 3, 1 Cycle of 1 leads to $\boldsymbol{\frac{p_2}{2} + p_3 = .25}$.

Therefore, the encoding scheme that achieves **optimal secrecy** is the encoding scheme in Figure 5, which has 2 cycles of size 2.

3.5 General Solution

In the above example, evaluating the possible optimal graphs was simple, because there were only two possible graphs. However, for larger sets of input source symbols, this problem is exponential in nature, because there are an exponential number of ways to tile the 2-cycles and 3-cycles. Thankfully, this problem can be solved in linear time by using a dynamic program that starts by optimizing a subset of the input source symbols with low probability, and increases the size of the subset until the entire set has been optimized. The code that generates this optimal scheme is provided in Listing 1.

4 lg(3)-Bit Secrecy

4.1 Problem Setup

Given the solution for the case in which we have one bit of secrecy, $\boldsymbol{k = 1}$, this section investigates how the general solution translates to more bits of secrecy. Before moving on to $\boldsymbol{k = 2}$, which would be 4 different possibilities given one encoding, the case where each encoding maps to 3 possible input sources is investigated. This case represents $\boldsymbol{k = lg(3)}$. A possible encoding scheme with 3 different secret key values is provided in Figure 10.

This graph is beginning to look very complicated. The dictionary book that would be inside the encoder and decoder is provided below in Figure 10.

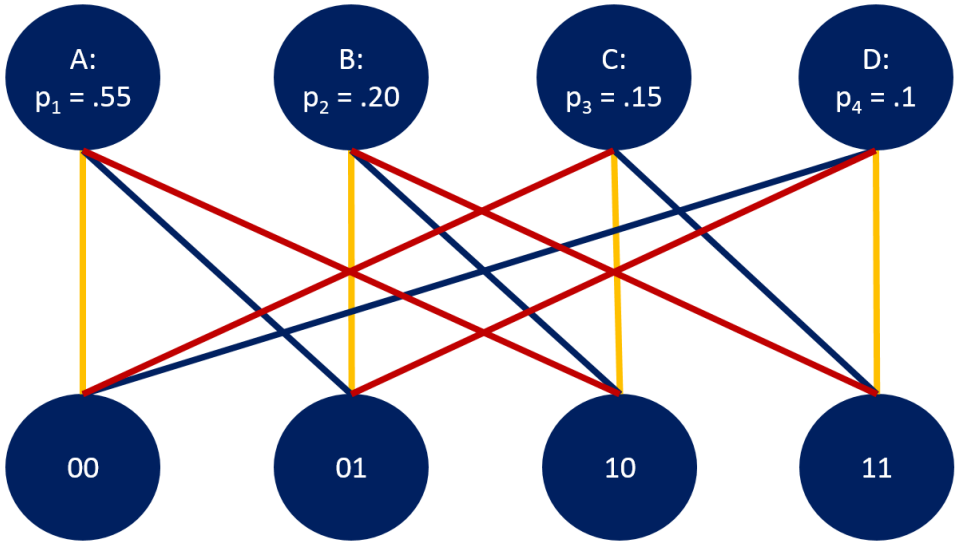

Figure 10: A complete graph with lg(3) bits of secrecy.

Source	Key = 0	Key = 1	Key = 2
A	00	01	10
B	01	10	11
C	10	11	00
D	11	00	01

4.2 Distortion

The eavesdropper's distortion is calculated just as before. The only difference is that the distortion on a particular bit pattern is not simply the smaller input source symbol divided by 2; rather, it is the sum of all probabilities that the eavesdropper does not guess divided by 3. This is demonstrated by the distortion values

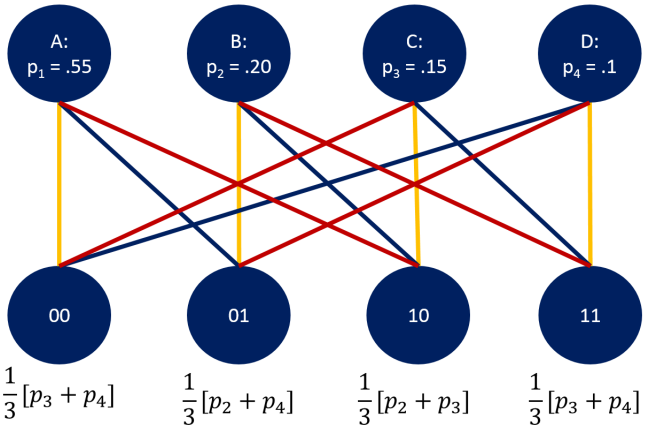

Figure 11: The distortion values associated with each bit pattern.

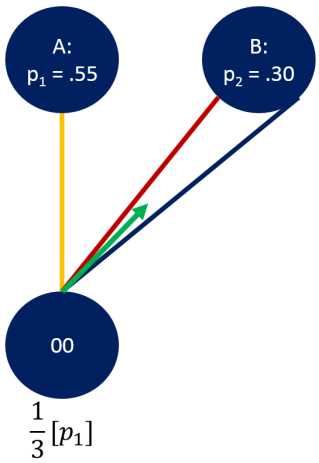

Figure 12: In this case the eavesdropper will guess B because $\boldsymbol{p_1 < 2p_2}$.

for each bit pattern provided in Figure 11. The total probability of error at the eavesdropper is $p_4+p_3+\frac{2*p_2}{3}$.

The interesting case that is introduced with $lg(3)$ bits of secrecy is when the eavesdropper will not select the most probable input source symbol that is connected to a bit pattern. This would occur if two key values point to an input source symbol that is slightly less than the most probable input source symbol. This case is demonstrated in Figure 12.

4.3 Hypothesis

The logical extension of the 1 bit of secrecy problem seems to be that there will be something similar to cycles that optimize the encoding scheme. The key point in the 1 bit of secrecy case was that the maximum size cycle was 3. Given this information, it seems reasonable that the largest size cycle in this case would be 5, and any cycle of size 6 would be split into two cycles of size 3. However, it is not clear whether a cycle of 5 input source symbols would form a graph that is in order. Either way, it seems reasonable that there is some limit to the size of a cycle when the amount of secrecy is $lg(3)$ bits.

4.4 Testing

To begin testing this hypothesis, a brute force solution is used to determine the optimal encoding schemes for sample probability distributions. This brute force approach takes$(|X|!)^2$ because all combinations of permutations must be evaluated. The brute force algorithm is provided in Listing 2.

4.5 Results

The brute force algorithm is tested with various input source probability distributions to determine the optimal encoding schemes with $lg(3)$ bits of secrecy.

Case 1: $X = A : .6, B : .3, C : .1$
Since this case has $lg(3)$ message bits, this encoding scheme should achieve **maximal secrecy**, and the probability of error should be $1 - p_{max}$. This is indeed the case. The encoding scheme uses the following dictionary:

Source	Key = 0	Key = 1	Key = 2
A	00	01	10
B	01	00	00
C	10	10	01

The probability of error is $p_2 + p_3 = 1 - p_{max} = .4$. Compared to the case in which we have 1 bit of secrecy which yields a probability of error of $p_2 = .3$, it is clear that the increase in secret key bits makes the encoding scheme more effective.

Case 2: $X = A : .3, B : .2, C : .17, D : .13, E : .12, F : .08$
The encoding scheme uses the following dictionary:

Source	Key = 0	Key = 1	Key = 2
A	000	010	001
B	001	000	010
C	010	001	000
D	011	101	100
E	100	011	101
F	101	100	011

The probability of error is $p_2 + p_3 + p_5 + p_6 = .57$. This case is an instance of the optimal encoding scheme being 2 cycles of 3. Therefore, the idea of cycles is still present in the $lg(3)$ case. However, this example does not provide any definitive general solution to the $lg(3)$ case.

Case 3: $X = A : .7, B : .11, C : .1, D : .06, E : .02, F : .01$
The encoding scheme uses the following dictionary:

Source	Key = 0	Key = 1	Key = 2
A	000	001	010
B	001	011	001
C	010	010	011
D	011	000	000
E	100	101	101
F	101	100	100

The probability of error is $\frac{2p_2}{3} + p_3 + p_4 + p_6 = .2433$. This case is an instance of the optimal encoding scheme being 1 cycle of 4 and 1 cycle of 2. The idea of cycles is still present in the $lg(3)$ case.

Case 4: $X = A : .6, B : .15, C : .09, D : .08, E : .07, F : .01$
The encoding scheme uses the following dictionary:

Source	Key = 0	Key = 1	Key = 2
A	000	001	010
B	001	000	000
C	010	100	011
D	011	010	100
E	100	011	001
F	101	101	101

The probability of error is $p_2 + \frac{p_3}{3} + p_4 + p_5 = .33$. This case is an instance of the optimal encoding scheme being 1 cycle of 5 and 1 cycle of 1. The idea of cycles is still present in the $lg(3)$ case. Given the original hypothesis, the assumption is that any cycle of 6 would be broken up into two cycles of 3 in order to increase the eavesdropper's probability of error. Therefore, if we find a scheme that uses a cycle of 6, this hypothesis will be proven incorrect. The next step is to try to prove this hypothesis false by finding a cycle of 6 that does not split up.

Case 5: $X = A : .5, B : .2, C : .15, D : .1, E : .03, F : .02$
This case disproves the original hypothesis because it contains a cycle of size 6. The encoding scheme uses the following dictionary:

Source	Key = 0	Key = 1	Key = 2
A	000	010	001
B	001	001	010
C	010	000	100
D	011	100	000
E	100	101	011
F	101	011	101

The probability of error is $p_2 + \frac{2}{3} * (p_3 + p_4) + p_5 + \frac{p_6}{3} = .4033$. This case disproves the hypothesis. In this case it is optimal to leave the encoding as a continuous graph, rather than break it into any disjoint cycles.

4.6 Further Research

The general solution to the $lg(3)$ case is yet to be determined. The research here provides a framework for testing probability distributions, and it shows the maximum cycle is 6 cycles or more. When the brute force algorithm was run on input sets with more than 6 source symbols, cycles continue to appear in the optimal

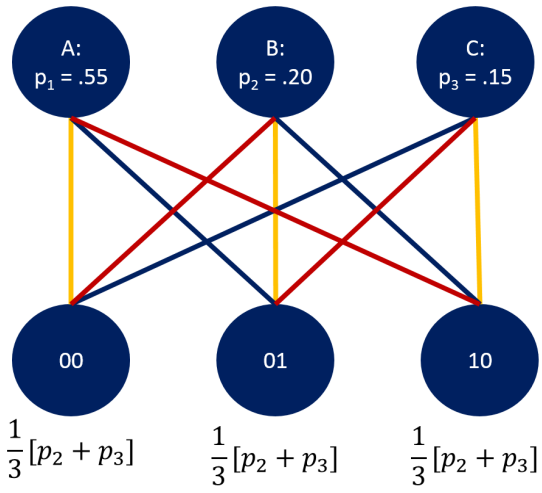

Figure 13: A 3-cycle with $\boldsymbol{lg(3)}$ bits of secrecy.

encoding scheme. However, even though it seems as though the concept of cycles will fit into this problem in some manner, the properties of these cycles still need to be determined.

4.7 Lower Bound

For the $\boldsymbol{lg(3)}$ case, a lower bound has been determined. For any set of input source symbols, the encoding scheme can be entirely cycles of 3, until the end of input source symbols where the cycle size can be 1 or 2, depending on the number of input source symbols. If the $|\boldsymbol{X}|$ is divisible by 3, the encoding scheme will be entirely 3-cycles.

A 3-cycle achieves the distortion demonstrated by Figure 13. The total distortion is $\boldsymbol{p_2 + p_3}$. Notice how this compares to the one bit of secrecy case $\boldsymbol{p_2 + p_3} > \frac{\boldsymbol{p_2}}{\boldsymbol{2}} + \boldsymbol{p_3}$. The lower bound for an encoding scheme with $\boldsymbol{lg(3)}$ bits of secrecy is:

$$\sum_{\boldsymbol{1 \leq i \leq |X|, i\%3 \neq 1}} \boldsymbol{p_i} \tag{2}$$

5 Stochastic Encoder

5.1 Problem Setup

If we increase the number of secret key bits, we can certainly begin to approach the upper bound given in Equation (1). However, the optimal scheme for more than 1 bit of secrecy is yet to be determined. Another means of possibly achieving **maximal secrecy** is by introducing a stochastic encoder. In this case there is not a 1:1 match between input source values and bit patterns. Now, there will be more encoding bit patterns than input source values.

5.2 Graph 1: Stochastic Process Split

In order to see the effects of introducing a stochastic encoder, Figure 14 is an encoding scheme with no stochastic encoder. In this case the probability of error for the eavesdropper is $\frac{\boldsymbol{p_2}}{\boldsymbol{2}} + \boldsymbol{p_3}$. This is actually the optimal encoding for this graph because $\frac{\boldsymbol{p_2}}{\boldsymbol{2}} + \boldsymbol{p_3} > \boldsymbol{p_2}$.

An example of a stochastic encoder for the same input source is provided in Figure 15. The intermediate nodes between the source symbols and the bit patterns are simply used as an analytic tool to calculate

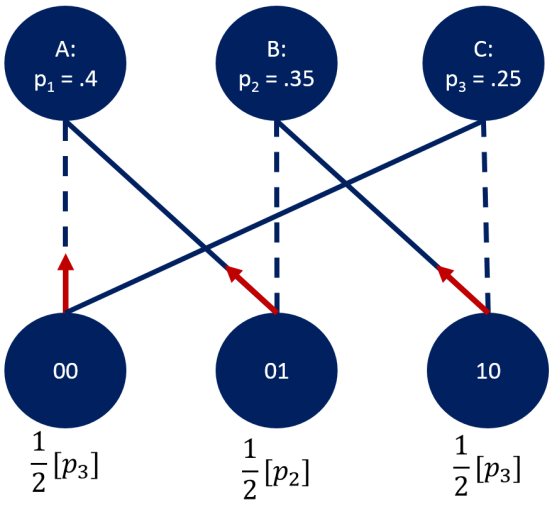

Figure 14: A 3-cycle with 1 bit of secrecy. The distortion introduced by each bit pattern is below each pattern. The eavesdropper's guessing pattern is represented by the arrows.

the distortion of this encoding scheme. In this case, the values of the intermediate nodes are constrained as follows: $\boldsymbol{X + Y = p_1}$ and $\boldsymbol{Z + W = p_2}$. In order for this constraint to be met, the intermediate nodes will take on the following values:
$\boldsymbol{Z = p_3}$
$\boldsymbol{W = p_2 - p_3}$
$\boldsymbol{Y = p_3}$
$\boldsymbol{X = p_1 - p_3}$

When the distortion of each bit pattern is simplified, the probability of error is: $\boldsymbol{p_2 + \frac{p_3}{2}}$. In comparison to a non-stochastic encoder, this scheme is superior: $\boldsymbol{p_2 + \frac{p_3}{2} > \frac{p_2}{2} + p_3}$. Numerically, the probability of error increases from **.425** to **.475** with the stochastic encoder. The **maximal secrecy** in this case would be **.5** because $\boldsymbol{.5 < 1 - p_{max} = .6}$. Therefore, this encoding scheme does not achieve **maximal secrecy**,

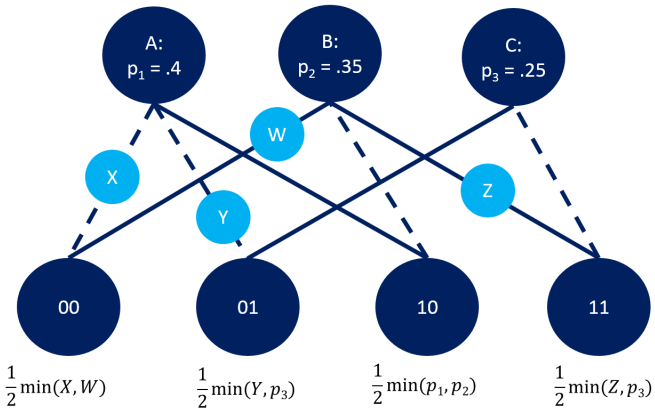

Figure 15: 3 messages with 4 bit patterns. The stochastic process is placed on the two most probable input source symbols.

but it does increase the probability of error.

5.3 Graph 2: Stochastic Process on One Symbol

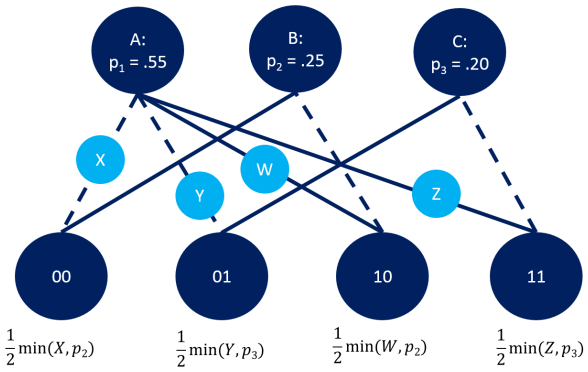

Figure 16: Stochastic encoder with the stochastic process is placed entirely on the most probable input source symbol.

In Figure 15, the stochastic process was placed on both the most probable and second most probable input source symbol. In addition to this possible scheme, the entire stochastic process could be placed on the most probable input source. An example of this encoding scheme is provided in Figure 16.

In this case, the values of the intermediate nodes are constrained as follows: $\boldsymbol{X+Y=p_1}$ and $\boldsymbol{Z+W=p_1}$. In order for this constraint to be met, the intermediate nodes will take on the following values:
$\boldsymbol{Z=p_3}$
$\boldsymbol{W=p_1-p_3}$
$\boldsymbol{Y=p_3}$
$\boldsymbol{X=p_1-p_3}$

When simplified, the total distortion becomes $\boldsymbol{min(p_{max}, 1-p_{max})}$. Therefore, for this example, with $\boldsymbol{p_{max}=.55}$, the encoding scheme achieves a probability of error of $\boldsymbol{1-p_{max}=.45}$. This encoding scheme achieves **maximal secrecy**. Therefore, with enough extra bit patterns used for encoding, the encoding scheme can achieve **maximal secrecy**.

5.4 3 Messages, 4 Encodings, 1 Bit of Secrecy

In summary, for the scenario in which 3 messages are encoded with 4 bit patterns and with 1 bit of secrecy, there are three cases:

Case 1: $\boldsymbol{p_{max} > p_2+p_3}$ The optimal encoding scheme applies all of the stochastic process to the most probable input source symbol. The probability of error is $\boldsymbol{1-p_{max}}$. This encoding scheme achieves **maximal secrecy**.
Case 2: $\boldsymbol{p_{max} < p_2+p_3}$ The optimal encoding scheme applies all of the stochastic process to the most probable input source symbol. The probability of error is $\boldsymbol{p_{max}}$. This encoding scheme does not achieve **maximal secrecy**.
Case 3: $\boldsymbol{p_{max} > p_2+p_3}$ The optimal encoding scheme applies the stochastic process to the most probable and second most probable input source symbol. The probability of error is $\boldsymbol{p_2+\frac{p_3}{2}}$. This encoding scheme does not achieve **maximal secrecy**.

5.5 Future Work

These cases completely solve the scenario described, however this is not a general solution. Future research should determine a vernal solution for the increase probability of error given the number of messages, number of encodings, and number of secret key bits. Some work has been done to study the case with 4 messages, 5 encodings, and 1 bit of secrecy, but a complete solution for that scenario has not been determined.

6 Variable Length Coding

6.1 Problem Setup

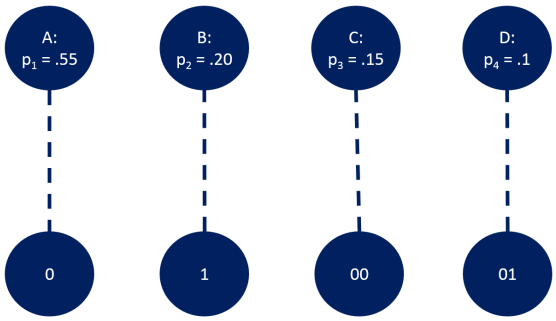

Figure 17: A variable length encoder with 0 bits of secrecy. The most probable source symbols have the smallest bit patterns.

In the proceeding discussion, all of the encoding schemes were fixed-length schemes . This is a brief consideration of the case in which variable length encoding is implemented. To simplify the discussion, organic punctuations between symbols are assumed; therefore, the encoding scheme does not need to be codes of prefixed length.

6.2 General Solution

The first step is to initialize the graph with 0 bits of secrecy. The encodings will be distributed according to the input source probability distribution. The most probable input source symbols will be encoded with the smallest possible bit pattern. This leads to a simple encoding scheme as demonstrated in Figure 17. Given this initial graph, in order to modify the graph to represent 1 bit of secrecy, solid lines must be added to the graph according to the encoding scheme for $\boldsymbol{S} = \mathbf{1}$. We know from Section 4 how to add the solid lines such that the probability of error is increased. This same logic can be followed in the variable-length case; however, if the average bit pattern length is the more important parameter, the encoding scheme would consist of only cycles that do not cross the boundary between bit patterns. For instance, the two most probable source symbols will always be encoded with a 2-cycle, because a 3-cycle will cross the boundary between a bit pattern of length 1 and length 2. Therefore, with one bit of secrecy the encoding scheme in the variable-length case will be very similar to the fixed-length case, except for some instances in which the algorithm will select the non optimal cycle in order to optimize the average bit length.

7 Conclusion

In cases where the amount of shared secrecy is minimal, certain encoding schemes are much more effective at increasing the eavesdropper's probability of error. One of the core characteristics of these encoding schemes is cycles. Grouping together input source symbols with similar probabilities increases the eavesdropper's probability of error. With one bit of secrecy, optimal encoding schemes can be determined in linear time.

For $\boldsymbol{lg(3)}$ bits of secrecy, a lower bound can be achieved in linear time, but the optimal encoding can currently only be found with a bruce force attempt, which take $\boldsymbol{|X|!^2}$ time. In order to achieve **maximal secrecy** a stochastic encoder can be used to increase the eavesdropper's probability of error. Additionally bit patterns used in the encoder will increase the eavesdropper's probability of error until the encoder achieves **maximal secrecy**. Lastly, for one bit of secrecy and variable length encodings, we can still use cycles in order to minimize the average bit pattern length.

A Source Code

A.1 onebitsec.py

```
#Authors: Sanket Satpathy and Mike Freyberger
#Sanket Satpathy: -onebitsec: dynamic algorithm
#Mike Freyberger: -printPath: used for debugging
#         -main loop: used for testing



import sys

def printPath(length, d, path, i):
  j = i
  numOfElements = 0
  while j >= 0:
    currentTupleSize = path[j]
    numOfElements+=currentTupleSize
    j-=currentTupleSize

  x = 1
  k = i
  j = numOfElements
  code =[]
  while x<numOfElements:
    if path[k]==2:
      code.insert(0,[j-1,j])
      x+=2
      j-=2
      k-=2
    else:
      code.insert(0,[j-2,j-1,j])
      x+=3
      j-=3
      k-=3

  k = numOfElements+1
  while k <= length:
    code.append([k])
    k+=1

  print(code),
  print("{0:.3f}".format(d[i]))



def onebitsec(p):
  # p is assumed to be in sorted order (descending)
  p=sorted(p,reverse=True)

  # list to store optimal distortions
  d=[p[1],p[1]*0.5+p[2]]+[0]*(len(p)-3)

  # list to retrace our steps up the tree of tilings
  path=[2,3]+[0]*(len(p)-3)

  #printPath(len(p), d, path, 0)
  #printPath(len(p), d, path, 1)

  # explore space of possible tilings
  i=len(d)-4
  for i in xrange(len(d)-3):
    if d[i+2]<d[i]+p[i+3]: # compare tilings that align
      d[i+2]=d[i]+p[i+3]
      path[i+2]=2 # 2-cycle
      #printPath(len(p), d, path, i+2)
```

```
    if d[i+3]<d[i]+p[i+3]*0.5+p[i+4]:
      d[i+3]=d[i]+p[i+3]*0.5+p[i+4]
      path[i+3]=3 # 3-cycle
      #printPath(len(p), d, path, i+3)

  i+=1
  if d[i+2]<d[i]+p[i+3]:
    d[i+2]=d[i]+p[i+3]
    path[i+2]=2
    #printPath(len(p), d, path, i+2)


  # find code corresponding to maximal distortion
  code=[]
  x=1
  j=len(p)
  if max(d[-1],d[-2])==d[-2]: # check if optimal code has 1-cycle
    code.insert(0,[j])
    x+=1
    j-=1
  # retrace steps
  while x<len(p):
    if path[-x]==2:
      code.insert(0,[j-1,j])
      x+=2
      j-=2
    else:
      code.insert(0,[j-2,j-1,j])
      x+=3
      j-=3

  # return optimal code and its distortion
  return [code,max(d[-1],d[-2])]

#Main loop that takes in a comma delimited pdf
#One pdf per line of input file
for line in sys.stdin:
  p = line.split(',')
  p = list(map(float, p))
  info = onebitsec(p)
  print info[0][:]
  print info[len(info)-1]
```

Listing 1: onebitsec.py

The probability distribution file is formatted as follows:

.3,.23,.17,.09,.08,.07,.04,.02

The probability distribution is saved in a file called *prob*, and the following unix command produces the optimal encoding scheme:

cat prob | python onebitsec.py

The output of this call is:

[[1, 2, 3], [4, 5, 6], [7, 8]]
0.415

This demonstrates the optimal encoding scheme for the probability distribution above is a 3-cycle, 3-cycle, 2-cycle, with the 2-cycle being on the least probable input source symbols.

A.2 multibitsec.py

```
#Author: Mike Freyberger

import numpy as np
import itertools as it
import sys

#general function for eavesdropper distortion
#this can handle any number of permutations.
#permutation: list of permutation arrays
#p: the original source probability distribution function
def calculateDistortion(permutation, p):
  # Determine the number of permutations.
  # This is correlated with the number of bits
  # of secrecy: bits = lg(numOfPermutations + 1)
  numOfPermutations = len(permutation)
  distortion = 0 #store the overall distortion
  distortionArray = []
  i = 0 #loop through the bit patterns
  while (i < len(p)):
    total = p[i] #store the sum of probablities connected to bit pattern
    j = 0 #loop through the symbols connected to this bit pattern
    max = p[i] #curren max
    decisionArray = {i: p[i]}
    # sum all the probablities and determine
    # the max input source symbol in the row
    while (j < len(permutation)): #loop through the symbols connected to this pattern
      currentLetter = permutation[j][i] #current symbol
      pPerm = p[currentLetter] #current prob
      if (currentLetter in decisionArray):
        decisionArray[currentLetter]+=pPerm #increment probability of symbol
      else:
        decisionArray[currentLetter] = pPerm #add symbol and probabilty to dict
      total += pPerm #sum all probablities
      if (decisionArray[currentLetter] > max ): #update max if necessary
        max  = decisionArray[currentLetter]
      j+=1

    currentDistortion = (total - max)/(numOfPermutations + 1)  #distortion of the row
    distortionArray.append(currentDistortion) #save current distortion
    distortion +=currentDistortion #update total distortion
    i+=1

  return {'array': distortionArray, 'total': distortion}


#Main loop that takes in a comma delimited pdf
#One pdf per line of input file
#Determines the optimal encoding with lg(3) bits of secrecy
for line in sys.stdin:
  probString = line.split(',')
  prob = list(map(float, probString))

  standardOrder = range(len(prob)) #ordering with no secrecy

  perms = [p for p in it.permutations(range(len(prob)))] #find all possible permutations

  i = 0
  j = 0
  max = {'total': 0}
  while( i < len(perms)): #loop through all permutations

    perms1 = perms[i]
    j = i #second loop will only look at combinations we haven't analyzed
    while (j < len(perms)): #loop through all permutations combintaions
      perms2 = perms[j]
      currentPermSet = [perms1, perms2]
```

```
67        dist = calculateDistortion(currentPermSet, prob)
68        if (dist['total'] >= max['total']-.00001): #account for floating point errors
69          max['total'] = dist['total'] #store new best permutation combination
70          max['array'] = dist['array']
71          max['perms1'] = perms1
72          max['perms2'] = perms2
73          #report = [prob, standardOrder, max['perms1'], max['perms2'], max['array']]
74          #display = np.array(report)
75          #print("Total Distortion is %.2f" % max['total'])
76          #print(display.T)
77        j+=1
78      i+=1
79
80    report = [prob, standardOrder, max['perms1'], max['perms2'], max['array']]
81    display = np.array(report)
82    print("Total Distortion is %f" % max['total'])
83    print(display.T)
```

Listing 2: multibitsec.py

This file can be used the same way onebitsec.py is used.