2020

# Musical Cryptography Using Long Short-Term Memory Networks

Curtis Helsel
*University of Central Florida*

# MUSICAL CRYPTOGRAPHY USING LONG SHORT-TERM MEMORY NETWORKS

by

CURTIS HELSEL

A thesis submitted in partial fulfillment of the requirements
for the Honors in the Major Program
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term, 2020

Thesis Chair: Richard Leinecker, Ph.D.

# ABSTRACT

Musical cryptography is a technique in which plain text messages are enciphered into a musical composition. Recently, a surge of music composition by means of machine learning have produced natural-sounding music that can be deemed as composed by humans. The combination of machine-generated music and enciphering a message into the composition is a logical step in musical cryptography. Outlined in this thesis is a method that incorporates the use of a specific type of recurrent neural network, Long Short-Term Memory, and a variant of the substitution cipher to form of symmetric-key cryptography system. Exploration was also completed to determine how the architecture of the network and the dataset used can modify the output of the encryption.

This thesis is dedicated to my wife and two daughters, who have supported me through countless hours of conversation on musical cryptography and machine learning musical composition.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

# INTRODUCTION

Cryptography is the process of enciphering and deciphering messages. In some cases, the act of communication is itself hidden as with the use of invisible ink, but more commonly, a message is masked with a cipher[1]. The key to unlocking the message contained within a cipher is known only to the parties which have access to the deciphering method. According to James L. Massey [2], cryptography has two main goals: secrecy and authenticity. The former ensures that the message being passed is decipherable only to the intended recipient, and the latter verifies that the recipient can authenticate the integrity of the message and identity of the sender. In classical cryptography, these goals were generally mutually inclusive. The secrecy of the message depended on being unreadable to anyone that was not aware of the deciphering mechanism, and authenticity was validated by the dependency that anyone without knowledge of the deciphering mechanism would not be able to create a system that could decipher it. It can be further stated that cryptography systems can be classified into either a symmetric-key system where a single key is used by both the recipient and the send or into an asymmetric-key system where a public key is known to everyone, and a recipient has a private key that they use for decryption [3].

Though recently popularized, machine learning and cryptography go hand in hand. In *Cryptography and Machine Learning,* Ronald Rivest explains that machine learning and cryptanalysis can be view as "sister fields" since they share similar tendencies [4]. In machine learning, the system being trained is analyzing a vast amount of data to determine patterns for understanding. A similar description could be given about cryptanalysts, searching for a pattern that allows them to reveal a secret message underneath. On the reverse side, researchers have

been able to create cryptographic systems that are entirely generated without the need for human intervention [5]. Martin Abadi and David G. Andersen demonstrated in their paper the possibility of training a network to learn to protect communications by providing a secrecy specification in the training objectives.

Cryptography is an evolving field that allows for the creation of new and more challenging to decipher methods of encrypting data. In this thesis, an approach for creating a novel method using recurrent neural networks to create a cryptographical system is outlined. The method is a form of symmetric musical cryptography where the sender and recipient will have access to a predictive model and a key for deciphering musical messages. In addition to the method of encryption, experimentation is done to determine how the architecture of the network and the dataset used to affect the output of the encryption.

## MUSICAL CRYPTOGRAPHY

Musical cryptography is not a new idea, and evidence shows that the earliest systems date as far back as the 15th century with many cryptologists have been notable musicians [1]. In Eric Sams' article, *Musical Cryptography*, he outlines numerous approaches of message encrypting with the most common method being that of assigning letters to individual notes of music. It was initially documented in *Tractatus varii medicinales,* the earliest system comprised of five different pitches, each with five varying note value and stem directions allowing for an alphabetic cipher of 25 characters. Giovanni Battista della Porta, known for Porta Cipher, created

a variation of the original cipher by using and 11 by 2 system shown in Figure 1. Famous Austrian composer, Johann Michael Haydn, built upon the Porta system with a variant that could be used in not only hidden communication but also in composing results that sounded of actual music. The substitution of notes to letter values continues to be a prevalent method for encrypting messages within musical scores.

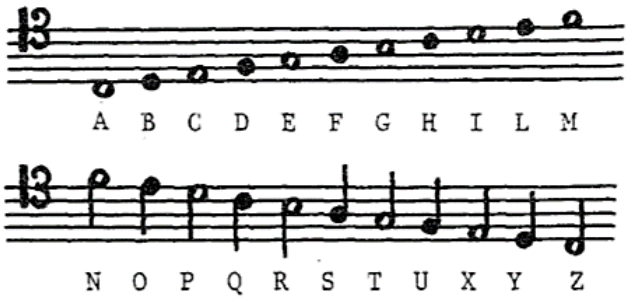


Figure 1: Musical Cipher
Source: Adapted from [1]

Modern musical cryptography has produced a number of novel methods for message hiding. Steganography, a comparable field to cryptography but differs in that cryptography protects the content of messages, while steganography is about concealing the message's very existence [6], has had many advancements in audio message hiding. Many of these systems exploits the masking effect property of the Human Auditory System (HAS) which allows the embedded information to be imperceptible [7].

One such method includes bit modification of the audio file. Due to the redundancy of bits in digital files, this technique is one of the most common and simplest methods for

embedding information. Replacing the least significant bit of the individual sampling points with a coded binary string allows a large amount of data to be encoded in the audio signal [8]. This method is not without it faults. Gopalan states any changes due to embedding would need to be small to prevent the detection of the secret information, or the change must occur at points that are masked by strong neighboring signals in the original host audio, for example, crowd noise [9]. This method also suffers from the vulnerability of lossy compression, filtering, amplification, and noise addition algorithms [7].

Another approach of audio steganography includes a technique created by Daniel Gruhl, Anthony Lu, and Walter Bender called Echo Hiding. This algorithm places the hidden text in the cover audio by introducing an echo. The message, in its binary representation, is added to the cover medium as a series of echo signals at specific delayed times. When these delay times are kept short, the two audio signals blend into a single distorted signal which is imperceptible to the human ear [10]. Gruhl et al. explain the encoding process as follows: the audio cover is divided into smaller sections and each of those sections can then be echoed with the desired bit. The stego audio is the recombination of all independently encoded signal portions. Decoding of the embedded text requires the detection of spacing between the echoes, which is done by examining the magnitude of the autocorrelation of the encoded signal's cepstrum.

In classical set theory, boundaries of a set are real-valued and can define whether an object belongs or does not belong to a set [11]. However, there are instances where this definition fails, e.g., hot, cold, short, and tall, and the use of a fuzzy set logic can be used. In terms of musical composition, Kumar et al. utilize this fuzzy logic to create musical sequences that are consonant. The authors generate a key matrix of probabilities from the seven possible

note combinations and produce candidate notes created from the rules in the fuzzy logic to encrypt the plaintext message into a musical sequence. The representations of candidate notes allow for a character to be encoded by many several candidate notes but not the reverse. Decryption is done by mapping the occurrence of notes from the key matrix back into a plain text message.

Another system using evolutionary computation was proposed by Kumar et al. in their article *Musical Cryptography using Genetic Algorithms.* Again using the transition probability matrix for the seven-note combination, they encrypted plain text messages by finding candidate notes for each character and applying a genetic algorithm for synthesis and sequencing of notes [12]. Every character will have a set of candidate notes, and the genetic algorithm will select the next possible note in the sequence based on a genesis rule defined for the transition of notes. This algorithm is applied over a set number of iterations to produce a musical piece that is pleasant to hear. Decryption is the same process of mapping the notes from the matrix to the characters.

## LONG SHORT-TERM MEMORY NETWORKS

Long Short-Term Memory networks, LSTM, are a type of gated recurrent neural network that uses historical sequence data to predict the next step in a sequence. These types of networks are prevalent in fields that analyze temporal data such as natural language processing or time series forecasting. A key characteristic that makes LSTM networks excellent for sequence learning is their ability to retain memory over long lengths of sequences. In standard recurrent

neural networks, as the distance between time steps grows, the error signals "flowing backward in time" tend to either grow exponentially causing the network to assign oscillating weights or decreases towards zero which produces a network that is either slow to learn or does not learn at all [13]. Hochreiter and Schmidhuber solved this issue by using a gradient-based algorithm that enforces a constant error flow through the internal states of the units.

An LSTM cell is comprised of several squashing functions, a cell state, and three gating mechanism: input, output, and forget. The squashing functions allow for the matrix data to remain within a specific value set. The hyperbolic tangent function has a range between negative one and positive one. The sigmoid function has a range between zero and positive one. As the data moves through the LSTM cells, the cell state is updated using the gating mechanisms. The forget gate takes the previous and new information, and using the sigmoid function determines which of the previous information should be kept [14]. Olah explains that the next gate used is the input gate, which determines which new information will be stored in the cell state. The final gate is the output gate, which is used to determine based on the information provided and kept what output should be sent to the next cell. It is the combination of these gates and the cell state the allow the LSTM networks to remember over long sequences of data.

## LSTM MUSIC GENERATION

A common method for generating sequences of musical notes is by using historical note inputs to generate the next note in the sequence [15]. As stated previously, recurrent neural

networks can retain memory over sequences of data which allows the prediction for musical notes based on what was previously seen. The LSTM networks outshine standard recurrent neural networks by being able to retain memory of sequences over long distances in time. Eck and Schmidhuber were able to use LSTM networks to train on a form of 12-bar blues used by jazz musicians. In their study, they showed that the network was able to produce new instances in the musical form by seeding musical notes into the prediction model and that recurrent neural networks can determine melody and structure of musical style. The implementation of LSTM musical generation is based on this process of training the network to look at historical note sequences and using seed notes from the data set to produce new musical melodies.

# MODEL IMPLEMENTATION

MIDI (Musical Instrument Digital Interface) is the standardized protocol for transmitting music control and timing information in real-time to hardware [16]. The file format that contains this information is used to create the sequential data that the network will train. In order to extract the information needed, the music21 library is used. The music21 project allows users to analyze, search, and transform music into a symbolic form [17]. The symbolic form retrieved from the midi file is in scientific pitch notation, shown in Figure 2, which is the representation of a musical pitch by concatenating the musical note (ABCDEFG) with an optional accidental (♯, ♭, ♮) and the octave which the note resides. Moog explains that the MIDI protocol provides a note capacity with a range of 128 possible notes, which spans more than 10 octaves. While this range is available as possible for notes in the training set, most musical notes fall within the standard 88 notes on a piano, which has a range from A0-C8. In addition to the symbolic form, each of the notes or chords will be appended with the duration of their quarter length. While the notes themselves should be able to produce a pleasing musical tone, the style of the dataset that is being used can be captured in time the notes are played.

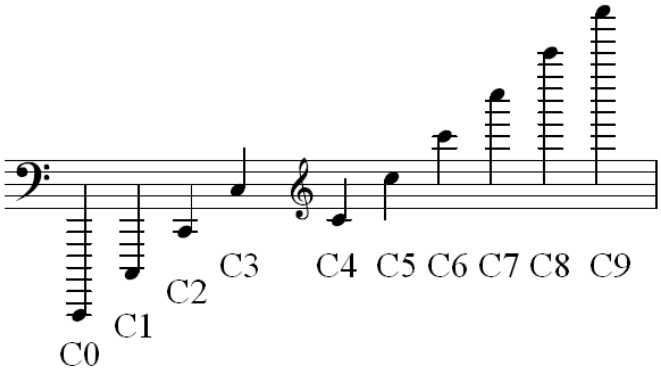


Figure 2: Scientific Pitch Notation of C Notes

Midi files may contain one or multiple tracks of musical notes. For each midi file in the data set, these tracks are broken down into their own individual sequences and parsed as if they were their own songs. This is done to ensure that not only the melody of the song was used for training but all parts as they pertain to the specific genre of music. The sequence of notes from each part read in from the midi file are divided into smaller sequences at a set length that is used to predict the next note in the sequence. A standard time signature for writing music is four beats per meter. The sequence lengths tested in the training were all multiples of this time signature: 4, 8, 16, 32. In the final architecture, a sequence of 16 notes is used to predict the next note in the song. Using sequences of 16 notes, a model was trained to produce an output based on any combination of 16 notes in the entire note set.

For the ease of use, the Keras library with the TensorFlow backend is used to create the predictive model. The determination for the architecture was done by modifying different components of the model to achieve a convergence of minimal loss within a reasonable number of epochs trained. The network is comprised of the following components: an LSTM layer with a 25 percent dropout rate on the output and a densely connected layer with the number of nodes equal to the number of different note values within the training data. The activation on the dense layer is the softmax activation, which allows for the prediction of the next value in the sequence. The loss for each iteration is calculated using categorical cross-entropy with RMSProp for the optimizer.

The data set used for creating the prediction model is comprised of 45 midi files of the genre American folk from a tune book website [18]. When broken down into the individual tracks, there are 191 instrument tracks with valid notes. The size of the entire note collection is

58,487, with 1709 unique note and chord values. Broken up into sequences of 16 for training, this becomes a data set of 58,479 data points.

For the use of this encryption algorithm, the validation on the training set was not used. While beneficial for musical generation to ensure the model is not creating the same tunes, overtraining or undertraining of the model is not a concern because the sequences are divided up into a note set for the use of encrypting and decrypting messages. If the resulting output for the encrypted message sounds similar to a piece of familiar music, this will only add to the level of secrecy of the hidden message.

# ENCRYPTION AND DECRYPTION ALGORITHM

Once a prediction model has been generated, it can be used to generate a note matrix from a note set to a character. As with the transition matrices in previously mentioned works, the matrix is used to encode and decode the messages. The message, once encrypted, is saved as a midi file that can be listened to on any music software that supports it. The decryption algorithm uses the midi file to reveal the hidden message. Both the sender and the receiver will have the pre-trained prediction model to encrypt and decrypt the hidden messages within the midi files.

## ENCRYPTION ALGORITHM

The encryption algorithm uses an encryption key, the prediction model, and the plain text message to produce a sequence of notes that are exported as a midi file. The encryption key is generated by using a pseudo-random number generator to select notes available in the dataset randomly. Using notes available in the data set allows the prediction model to generate a sequence of notes used for the note matrix. The number of notes predicted will depend on the length of the alphabet being used and the number of notes per character in the note matrix. The number of notes per character is provided by the user at the time of encryption. This allows for varying lengths of melodies as well as to provide an additional variable to make deciphering the message more difficult for those without the key. For this thesis, the 100 printable characters from Python's string library, shown in Table 1, are used for encoding and decoding the message. The notes per character have been set to 4.

Table 1: Python String Constants

| String Constants | Characters |
|---|---|
| string.ascii_lowercase | abcdefghijklmnopqrstuvwxyz |
| string.ascii_uppercase | ABCDEFGHIJKLMNOPQRSTUVWXYZ |
| string.digits | 0123456789 |
| string.punctuation | !"#$%&'()*+,-./:; |
| string.whitespace | space, tab, linefeed, return, formfeed, and vertical tab |

Once the set of notes has been generated and characters assigned to the note sets, the message can be encoded by selecting the note sets that correspond to the character in the message to be encrypted. This is a form of a substitution cipher where there is a one to one relationship between the note sets generated and their respective characters. The set of notes is exported to the midi format, and the encryption key is placed in the metadata of the file for retrieval when being decrypted. This allows the receiver only to receive one file for the decryption. The complete encryption process is shown in Figure 3.

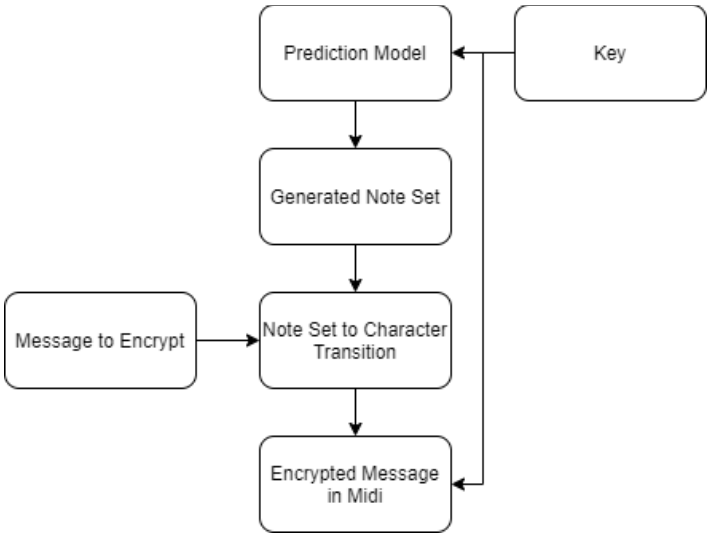


Figure 3: Encryption Process

## DECRYPTION ALGORITHM

The decryption algorithm reverses the encryption by using an encryption key, the prediction model, and the midi file to produce the plain text message. The decryption has a similar process to encryption. The key is extracted from the metadata of the midi file and is used as a sequence of notes for the predictive model to generate the note matrix. The matrix is divided into the note sets of the length provided in the key. The notes from the midi file are parsed using the note matrix to convert the notes back into characters. The characters are the plain text message that was encrypted. The complete encryption process is shown in Figure 4.

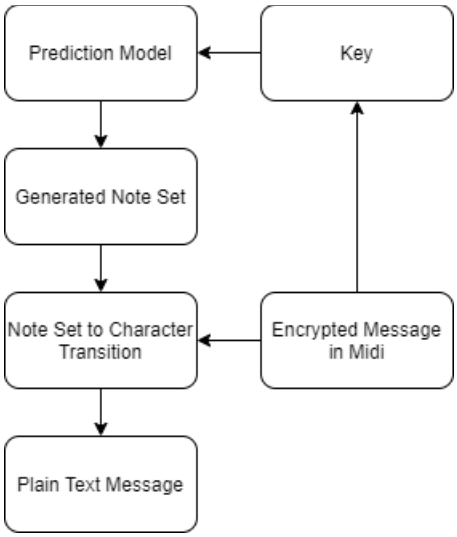


Figure 4: Decryption Process

# EXPERIMENTATION

In addition to creating a process for using long short-term memory networks in musical cryptography, two experiments were performed to see how the generated sequences used in the note matrix would be affected by the change in the architecture and dataset. The first of these experiments look at the change in sequence prediction over iterations compared to the final iteration. The second compares predictions based on removing portions of the dataset.

## CHANGE OF PREDICTION OVER EPOCHS

As the predictive model goes through the training process, the predictions on provided sequences will change based on the loss for each epoch. This means that there could be a possibility that at some point, as the model moves towards the lowest point in gradient descent, sequences will predict the same notes. In normal music generation, this process would be beneficial as it means that the system has learned to produce music. In the cryptographic sense, this would be detrimental if someone, for which the message is not intended, had information about the architecture (number of layers, LSTM units, and batch size), the data set, and the midi file containing the secret message. They would only need to train the model until it no longer had a reduction in loss.

The comparison of prediction was implemented using the following architecture:

- 57,186 16-note sequences.

- 1 LSTM layer with 256 units.
- 1 Densely connected layer with nodes equal to the size of the note set.
- Batch size of 256.
- 1000 iterations of training.

From Figure 5, the loss of the model steadily decreased through the first 300 iterations and a small amount through the remaining 700 iterations. The loss from iteration 300 to 1000 changes 0.1457, and in the last 150 iterations, the change is 0.0064. If the model is converging around a single loss, then that should mean that the model created in the final epoch should not be mostly different from the model in the previous epochs.

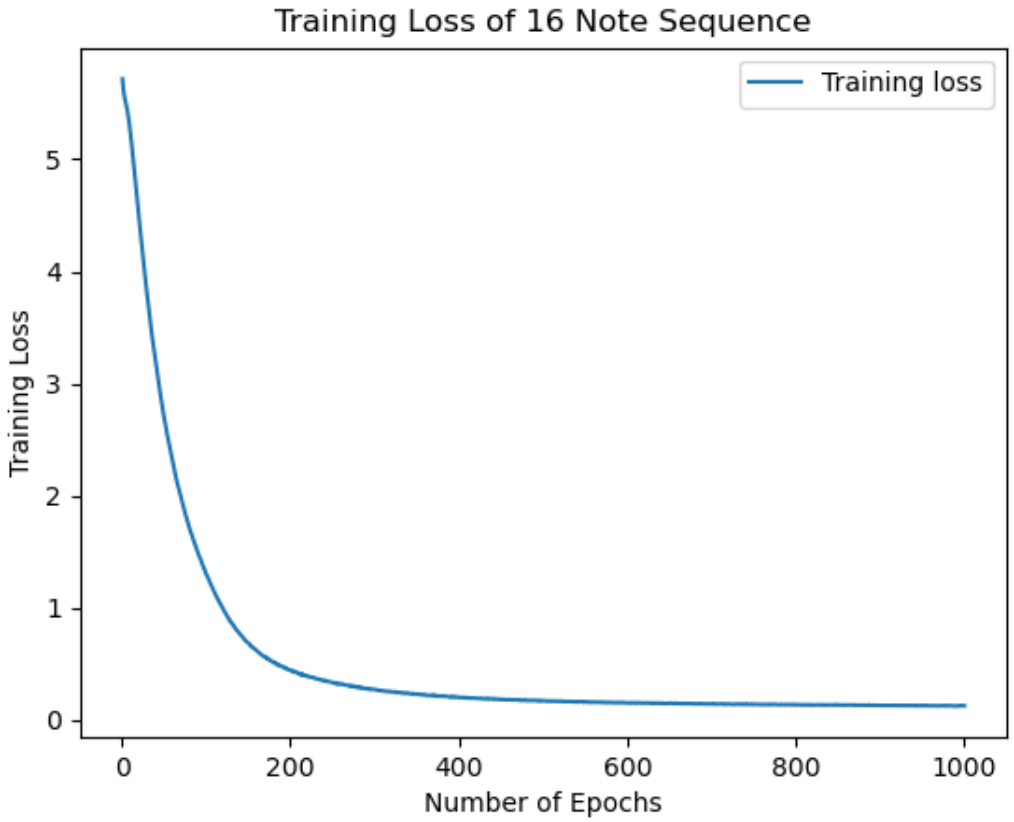


Figure 5: Training Loss of Prediction Model

For each epoch, a model is saved so it can be used to compare predictions against the final epoch. For the comparison, 1000 unique keys are generated on the note set, and the number of same predictions for the test and final epochs is counted towards a total percent match between the models. In Figure 6, a gradual increase in percent match is seen as the test epoch gets closer to the final epoch, however, even with a small loss difference, the percent of predictions that matches the number of predictions is less than 50 percent.

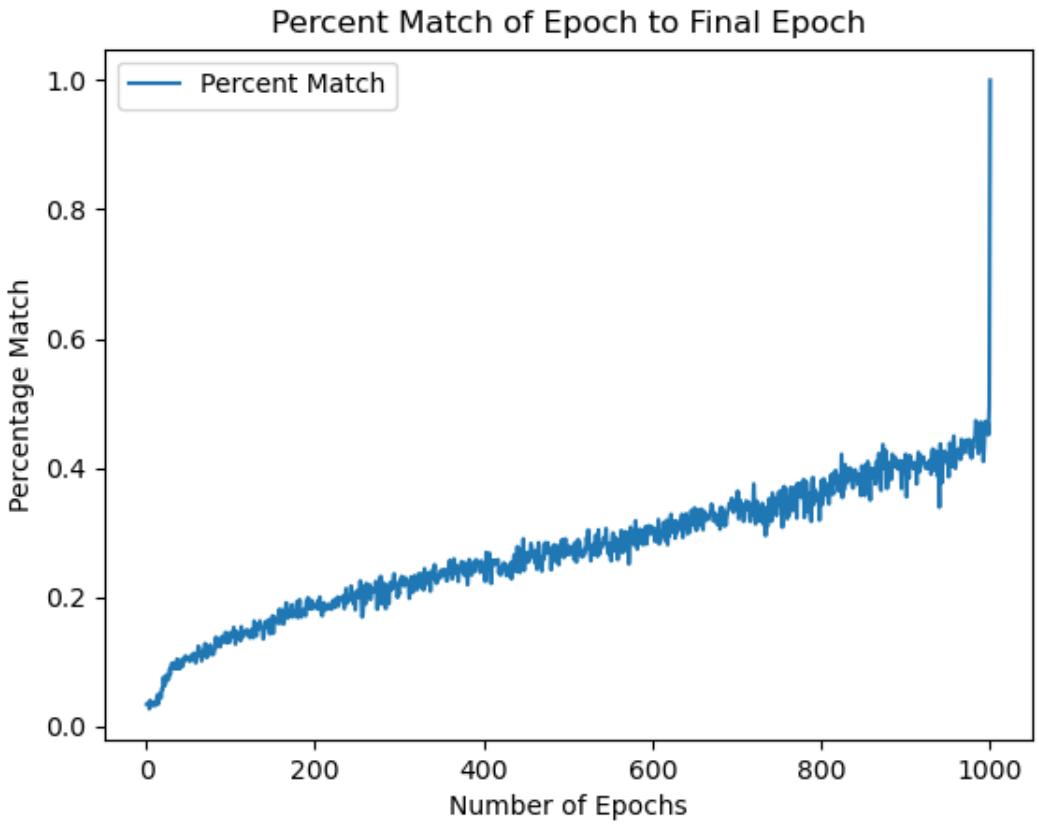


Figure 6: Percent Match Prediction - Single Output

This shows that if someone were to train for many epochs, they would not be able to get the same prediction 50 percent of the time. The percentage drops at the length of the prediction sequence increases. A sequence of 26 notes, which corresponds to the number of letters in the English alphabet, reduces the percentage to less than 15 percent for the next closest epoch.

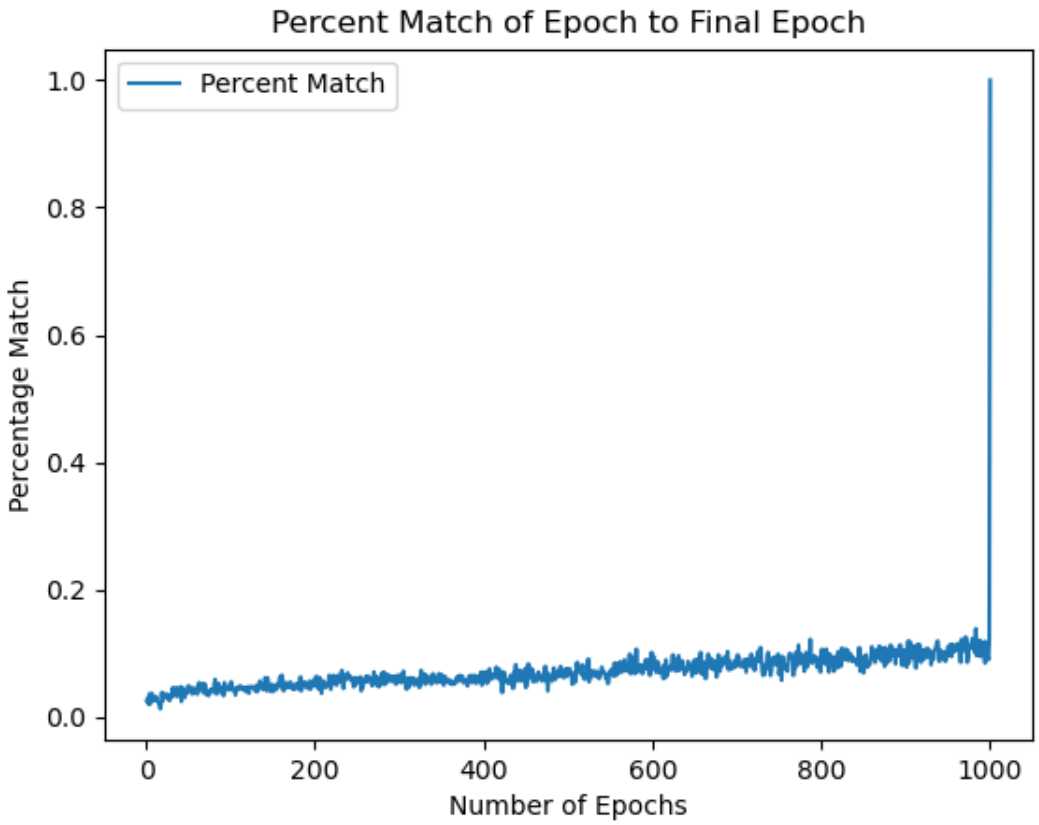


Figure 7: Percent Match for 26 Note Sequence

An additional point can be made that if the model was trained to an epoch that did not have a minimal loss, then training a model to the point of having minimal loss will not increase the chance of matching predictions. Having trained a model for only 200 epochs would have a difference of 80 percent from a 1000 epoch minimal loss model. This beneficial from the standpoint of cryptography as it decreases the likelihood of being able to brute force the prediction to produce a valid note matrix.

Training on a full data set provides a model that should be able to predict sequences that are contained within that data set. A subset of this data set contains the same sequences as the full data set, which should mean that a model trained of the full data set should have the same predictions as a model that is a subset for sequences that are contained within both sets. If this were true, a note matrix created for encrypting messages would be at risk for exposure if a model with minimal loss is trained on a data set that encompasses the data set used to produce the model.

In this experiment, the full set is reduced to subsets of 25 percent, 50 percent, and 75 percent of the original set. With the same architecture as above but instead using a sequence length of 8 notes for 200 epochs. After 200 epochs, the loss is similar between the four models.

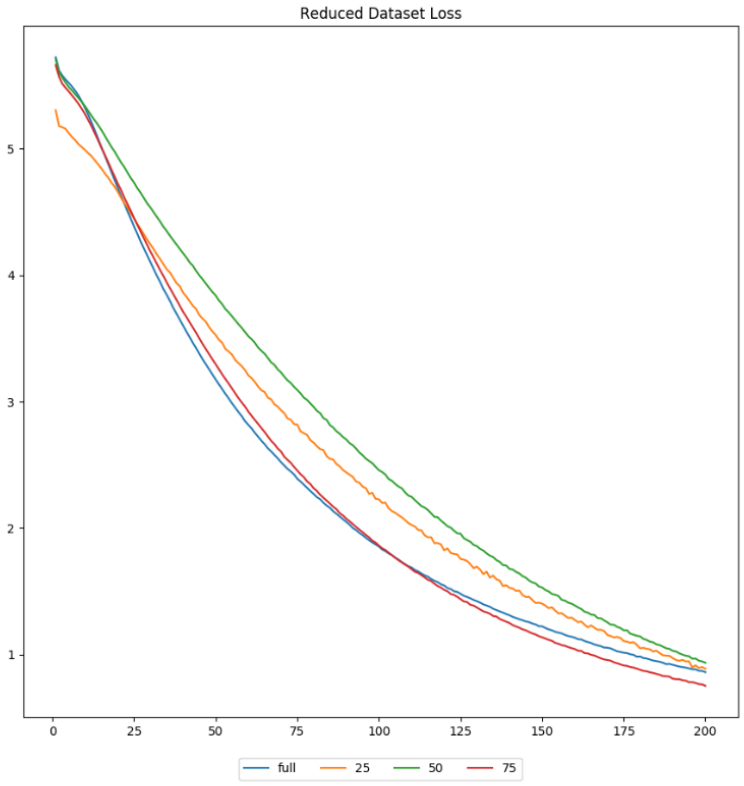


Figure 8: Reduced Dataset Loss

On the three reduced models, 1000 keys were generated that ensured that the notes would be contained in both the test model and the final model. A comparison for the prediction accuracy, in Figure 9, between the two models, shows that as the subset size decreases, the level of percent accuracy of the full set model increases. However, the prediction accuracy never exceeds 5 percent, which means having a data set that encompasses a smaller set does not mean that the model will be able to decrypt the hidden message on the smaller training set note matrix.

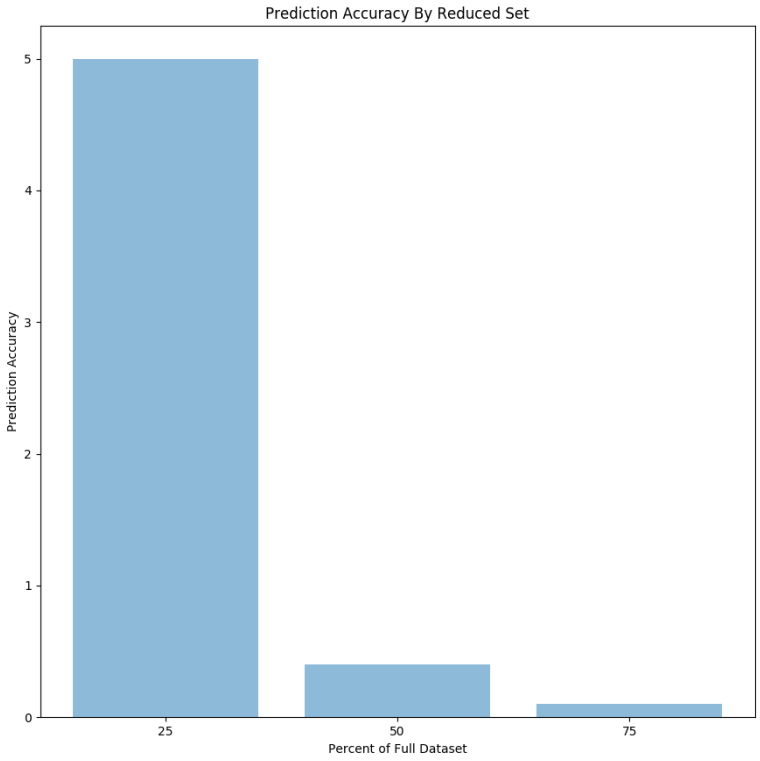


Figure 9: Prediction Accuracy for Reduced Sets

# CONCLUSION AND FUTURE WORK

Similar to most substitution algorithms for cryptography, the method outlined in this thesis is subject to statistical analysis of letter frequency. The method would be less susceptible to attack if a smaller cryptographic message is created and by using a newly generated key for each message. A more robust approach could be to link more recurring characters to additional sets of notes allowing for a one-to-many relationship that would make frequency analysis difficult.

One issue with the algorithm is that while it does produce a musical melody, not all melodies, it produces sound pleasant. This is due to the generation creating a sequence that is a musical output in one continuous form. Taking portions of that sequence and arranging them in different ways does not always work out. However, the benefit of the random key generation is that a new note matrix can be created to form better-sounding melodies. A possible remedy for this issue is to create a note matrix tree that, given any specific character, would build several branches with note sets that correspond to the next character. This would ensure that the message being encrypted has an accompanying tune that is pleasant to hear.

Despite the above issue, using long short-term memory networks to build a cryptographic system could be beneficial. The number of parameters that can be set within a network and the varying sizes and types of data show that it would be very cumbersome to attempt to decrypt.

# REFERENCES


[1] E. Sams, "MUSICAL CRYPTOGRAPHY," *Cryptologia,* vol. 3, no. 4, pp. 193-201, 1979/10/01 1979, doi: 10.1080/0161-117991854052.

[2] J. L. Massey, "Cryptography—A selective survey," *Digital Communications,* vol. 85, pp. 3-25, 1986.

[3] A. Joseph and V. Sundaram, "Cryptography and steganography–A survey," 2011.

[4] R. L. Rivest, "Cryptography and machine learning," 1991: Springer, pp. 427-439.

[5] M. Abadi and D. G. Andersen, "Learning to protect communications with adversarial neural cryptography," *arXiv preprint arXiv:1610.06918,* 2016.

[6] F. A. P. Petitcolas, R. J. Anderson, and M. G. Kuhn, "Information hiding-a survey," *Proceedings of the IEEE,* vol. 87, no. 7, pp. 1062-1078, 1999.

[7] F. Djebbar, B. Ayad, H. Hamam, and K. Abed-Meraim, "A view on latest audio steganography techniques," 2011: IEEE, pp. 409-414.

[8] W. Bender, D. Gruhl, N. Morimoto, and A. Lu, "Techniques for data hiding," *IBM systems journal,* vol. 35, no. 3.4, pp. 313-336, 1996.

[9] K. Gopalan, "Audio steganography using bit modification," 2003, vol. 1: IEEE, pp. I-629.

[10] D. Gruhl, A. Lu, and W. Bender, "Echo hiding," 1996: Springer, pp. 295-315.

[11] C. Kumar, S. Dutta, and S. Chakraborty, "Hiding messages using musical notes: A fuzzy logic approach," *International Journal of Security and Its Applications,* vol. 9, no. 1, pp. 237-248, 2015.

[12] C. Kumar, S. Dutta, and S. Chakborty, "Musical cryptography using genetic algorithm," in *2014 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2014]*, 20-21 March 2014 2014, pp. 1742-1747, doi: 10.1109/ICCPCT.2014.7054851.

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735-1780, 1997.

[14] C. Olah, "Understanding LSTM Networks," vol. 2019, ed, 2015.

[15] D. Eck and J. Schmidhuber, "A First Look at Music Composition using LSTM Recurrent Neural Networks," Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale, 2002.

[16] R. A. Moog, "MIDI: musical instrument digital interface," *Journal of the Audio Engineering Society*, vol. 34, no. 5, pp. 394-404, 1986.

[17] M. S. Cuthbert and C. Ariza, "music21: A toolkit for computer-aided musicology and symbolic music data," 2010.

[18] B. Taylor. "Traditional American Tunes in Midi Format." https://www.contemplator.com/tunebook/america.htm (accessed October 25, 2019).