Decrypting Ciphertext Using Deep Learning

Omar Einea
u18103996@sharjah.ac.ae
University of Sharjah

Abstract – Technological advancement is a two-edged sword, it could become a threat to our security as we build more intelligent systems that might break it. This research explores the potential of using such systems to counter our security measurements, by using Deep Neural Networks to perform a chosen-ciphertext attack to extract useful information from a ciphertext. We introduce a Deep Learning model that could be used to evaluate the security of any encryption function, and we show and compare the results of running it on several ciphers.

Index Terms – Decryption, Deep Learning, Machine Learning, Neural Networks

INTRODUCTION

Cryptography is an ever-advancing field which's essential in our daily lives, it's needed to ensure the security of stored information as well as the security of transferred data. That was done since thousands of years through encryption algorithms that deforms the sent message (plaintext), making it not readable (ciphertext) to anyone who doesn't know how to interpret (decrypt) it. These early methods are usually referred to as classical encryption, and since then many algorithms have been proposed year after year. Each time someone introduces a new algorithm, another comes to break it and prove its insecurity.

Nowadays, we're settled with what we call modern encryption, we use it to secure the data on our devices, servers and to secure the communication among them. We take it for granted that they are unbreakable, but that might change due to the breakthroughs and advancements of technology.

On the hardware level, we now have the ability to do intensive processing on super computers, which was actually used by Google to break the wildly used SHA1 hash function [1] and show the possibility of collisions happening when using it. This took them around 6500 CPU years and 100 GPU years, which's unheard of before.

On the software level we even bigger revolutions with the power of Artificial Intelligence, and especially Machine Learning. Machine Learning is excellent at finding patterns between a given input and an expected output without having to explicitly program it to do so. In theory, it should be usable to recognize patterns to deduce the original text from ciphertext to some extent. But modern encryption algorithms were designed to be statically random, leaving no trace for a pattern or a link to be found between plaintext and ciphertext.

In this paper, we examine the extent to which Machine Learning can be utilized to decrypt ciphertext. We used Deep Neural Networks to build the architecture of our model and train it on (plain, cipher) text pairs. As an outcome of this research, we propose a generic model which can be used to test the security of any given encryption algorithm, and we show the results of that model on a collection of famous encryption models.

RELATED WORK

There aren't many works in this particular research area because of the complexity and near impossibility of cracking modern encryption algorithms. But still some researchers gave it a go.

The closest paper to this work is one where they attempted and succeeded at decrypting DES and Triple DES algorithms [2] using a neural network to retrieve the plaintext from ciphertext without knowing what the key used for encryption is. Similarly, in [3] they presented a Neural-Identifier and used against Simplified DES to determine the key from given (plain, cipher) text pairs. Also, there's the work done in [4] where they tested Artificial Neural Networks as a Cryptanalysis tool against DES encryption.

In another work [5], they show the ability of Deep Learning at performing side-channel key recovery attacks, focusing on temples and concluding that Deep Learning based attacks are more efficient than other Machine Learning based one. They tested multiple Machine Learning approaches on masked or unprotected cryptographic implementations.

This paper [6] display a successful attempt at extracting targeted bytes of keys use with masked AES encryption, and they show that their method is more efficient than previous attempts. Similarly, in [7] they used Neural Networks based attack against masked AES to identify the mask to weaken the encryption.

Another attempt [8] shows the use of Support Vector Machine to perform a profiling-based side channel attack to recover a cryptographic secret. In [9], they used an adaptive multi-class SVM to reduce the effort needed to carry out the attack. And very recently in [10], they suggested another improvement over SVM making it require fewer side-channel attack power traces to recover the key.

Another related work is [11], as a first step to breaking the encryption, they used SVM to identify the block cipher encryption function used only from ciphertext.

IMPLEMENTATION

In this section, we describe the structure of the model we used to perform two types of attacks on several cipher algorithms:

Model:
As mentioned before, we used Machine Learning to build our model, a namely Deep Neural Networks model. The reason why we chose Deep Learning is because it outperforms other Machine Learning approaches as long as there's large enough data to train on. And in the case of this research, generating large amounts of data is an easy task as it only requires a list of randomly generated plaintext and its corresponding list of ciphertext.

Following is the structure of our model:

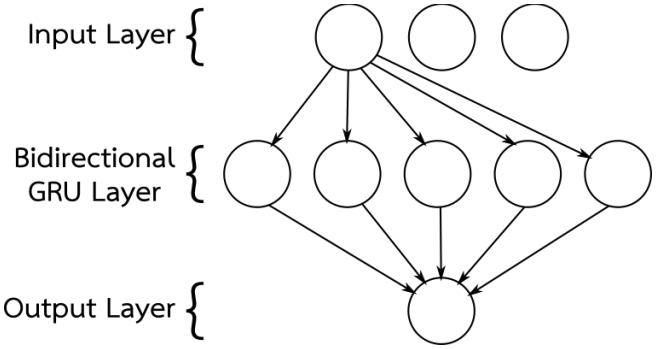

Figure 1: Model Structure

1. *Input layer:* A simple layer that accepts a ciphertext (feature) represented as a vector of integer values, each integer represents a character (much like the ascii representation of characters). The layer then maps those vector values to the next layer.
2. *Bidirectional GRU Layer(s):* Gated Recurrent Unit (GRU) is a Recurrent Neural Network layer that understands context and relationship between word or characters. Bidirectional GRU is GRU that trains on text in both directions, this was essential to crack shifting or transposition ciphers.
3. *Output Layer:* A Dense layer that maps the values coming from all neurons of the previous layer into one single vector representing the plaintext or secret key (label) which corresponds to the inputted ciphertext.

The library used to build and train the model is the well-known Keras library using TensorFlow underneath, it is a robust framework with an easy to use interface.
To represent textual data in a format that's understood by the model, we used the Tokenizer interface provided by Keras to convert text into integer vector representation.

We also used Early Stopping interface from Keras to stop the training process whenever it doesn't improve for a number of consecutive iterations (epochs). That prevents the model from overfitting on the data and saves energy and time by stopping the expensive training process earlier.

Attacks:
In this work, we examined two different attacks using the same model with a slight change to adapt to each of the attacks. These attacks are two variations of the chosen-ciphertext attack, which is about extracting useful information from ciphertext chosen (generated) by us. The two attacks are as follows:

I. Retrieving plaintext from cipher:
By feeding the model a list of ciphertexts as the input and its corresponding list of plaintexts without any knowledge of the secret key, the model learns to correlate between ciphertext (feature) and plaintext (label) such that once trained, the model should be able to predict the plaintext when provided with a ciphertext it hasn't seen before.

II. Retrieving secret key from ciphers:
In this attack the model doesn't have any knowledge of the plaintext. Rather it is provided with a set of examples, each example is a list of ciphertexts and the key they were encrypted with. Each example has a different key, and the model's role is to learn to correlate between each ciphertext list (feature) and their key (label) such that when provided with a list of ciphertext, the model should predict the key they were encrypted with.

Ciphers:
We used several encryption functions, some are classical (implemented by us) and few are modern (from a library). The cipher functions are as listed below:

Classical Encryption:

- *Caesar Cipher:*
  most basic one where each letter is shifted by 3 as a default value, but it can be specified to be any other value.
- *Vigenère Cipher:*
  each letter is shifted by the alphabetical index of the letter that corresponds to it from the secret key. If no key is specified, the shifting becomes linearly incremental for each latter.
- *Rotating Cipher:*
  rotates the plaintext after the given value. Rotation is performed by simply splitting the plaintext at the index of the giving value and switching the two part with one another. This cipher is used to make other ciphers slightly more complex.
- *Transposition Cipher:*
  a very simple one iteration implementation where the key is a map that repositions each letter in the plaintext to a new location (no change of letters, just the position).
- *XOR Cipher:*
  performs bitwise XOR operation between each letter from the plaintext with the corresponding letter from the secret key.

Modern Encryption:

- *DES Cipher:*
  Data Encryption Standard (DES) is a symmetric-key block cipher which uses 16 round Feistel structure. It is based on fixed size blocks of 64 bits, and the key has the same size as well.
- *AES Cipher:*
  Advanced Encryption Standard (AES) is a more advanced symmetric-key block cipher based on fixed size block of 128 bits, but the key can be of size 128, 192 or 256 bits.
- *Fernet Cipher:*
  Builds on best practice encryption functions. It uses 128-bit AES symmetric encryption in a CBC mode with padding, and HMAC using SHA256 for authentication.

EXPERIMENTS

As any Machine Learning experiment, ours is split into two phases, a training phase and a prediction phase after the model has been trained. In training the model finds relationship between ciphertext and plaintext, and in prediction it tries to estimate the plaintext that corresponds to a new given ciphertext.

We have two experiments for the two attacks, and we've tested each of them with a subset of the cipher methods listed above. First, we'll detail retrieving plaintext from cipher attack thoroughly, then we'll briefly mention the different between it and retrieving secret key from ciphertext attack as they're very similar.

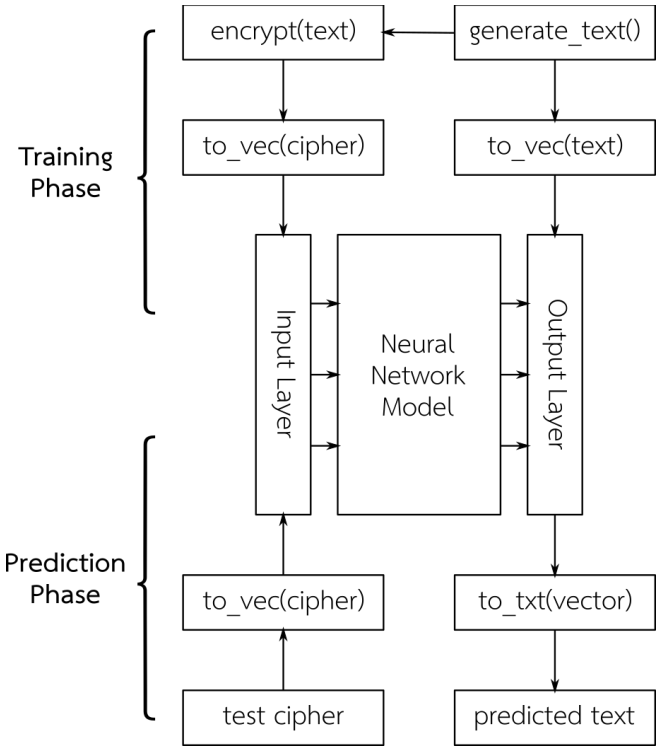

Figure 2: Training and Prediction Workflow

Retrieving plaintext from cipher:

1. *Generating Plaintext:*
   Using a simple function of ours, we specify the characters set, text length the number of samples and it generates a list of plaintexts based on the specified parameters. In most of our experiments, those parameters were set as:
   - character = alphabets and digits (62 character)
   - text length = 10 or 8
   - number of samples = 10^5
2. *Producing Ciphertext:*
   By applying one of the cipher functions listed in the implementation section on the generated plaintext list, we get another list of ciphertexts. Each item in that list is the ciphertext of the corresponding plaintext in the plaintext list using the chosen cipher function.
3. *Vectorizing Both Texts:*
   Keras has a Tokanizer interface which we fit on the specified characters set so that it memorizes a unique integer value for each character. Then we use it to tokenize (i.e. convert to integer vector) both plaintext list and ciphertext list to have them ready for the Machine Learning model to understand them.
4. *Building Model Structure:*
   Here we compile the model structure mentioned in implementation section (Figure 1), which consists of an input layer, a Bidirectional GRU layer with 128 neurons and an activation function called RELU which replaces negative values with 0. And lastly a Dense layer with one neuron which corresponds to the single output (plaintext).
5. *Training the Model:*
   Start training the model after feeding it both vectorized lists of (plain, cipher) text. Here we specify the following parameters for training:
   - Loss function = Mean Squared Error
   - Optimizer = Adam
   - Validation Split = 10%
   - Number of Epochs = 10
   - Batch Size = 10
6. *Predicting on the Model:*
   Once the training is done, we could see the accuracy in the logs. But to make sure everything is correct, we perform manual testing by sending the model a vectorized test cipher to do the prediction. The model would return a vector representation of the predicted plaintext, so we need to convert it to text before comparing it with the real one.

Retrieving plaintext from cipher:

Following the same steps above except for these differences:

For training, we generate plaintext once and generate a list of keys, for each key we generate ciphertext list then feed the keys list and the list of ciphertext lists to the model and train.

For prediction, we use a new key, generate ciphertext list for it, predict on that list and compare the returned key (after converting it to text) with the original one.

RESULTS AND DISCUSSION

Below is a list of our models' prediction accuracy results after performing various tests using the workflow mentioned in the experiments section (Figure 2).

Results for Retrieving plaintext from cipher attack:

Encryption Function	Prediction Accuracy
Caesar Cipher	100%
Vigenère Cipher	100%
Shifting Cipher	100%
Transposition Cipher	100%
Shifted Vigenère Cipher	97.5%
Keyed Vigenère Cipher	97.1%
XOR Cipher	52.6%
Fernet Cipher	10.1% (padding match)
DES Cipher	8.3% (padding match)
AES Cipher	8.1% (padding match)

Results for Retrieving plaintext from cipher attack:

Encryption Function	Prediction Accuracy
Transposition Cipher	80.8%
Vigenère Cipher	58.3%
DES Cipher	0%

Looking at the results, it is clear that classical ciphers are breakable as they it is easy for the model to link between the ciphertext and plaintext. But when it comes to modern encryption, the model failed at finding any relation because of the randomness of the cipher. And even this small percentage of accuracy was mostly due to padding and encoding characters match, not an actual match of characters that belong to the original plaintext.

The second attack is more useful that the first one because it predicts the key rather than the plaintext, but it seems to be much harder to perform even on classical encryption let alone the modern ones.

Most results are as expected and came out with not much of a surprise, what was interesting though is the low accuracy of predicting on some classical encryption (like XOR) which don't contain any element of randomness and patterns should be easily detectable in theory.

FUTURE DIRECTIONS

This research area needs more attention. The work done in this paper is an attempt to create a general decryption model that works on any cipher function, but we believe that a more focused model that's tolerant to a specific kind of encryption or better yet to one single implementation of a cipher function would yield better results.

No encryption is perfect and there's always room for improvement, it's only because of the work of the people who came before us that we live in the secure world we take for granted today.

CONCLUSION

In this paper, we proposed a general Neural Network model to evaluate the security of any encryption function. We showed that Deep Learning could crack classical encryptions easily, but it fails to do so for modern encryption methods. We discussed that the reason for this is that modern methods were designed to be statically random leaving no detectable patterns between ciphertext and plaintext. We believe that more targeted and fine-tuned Neural Network models, could be developed in the future to give better results and indicate the security level of a given encryption algorithm with higher accuracy.

REFERENCES

[1] Stevens, M., Bursztein, E., Karpman, P., Albertini, A., & Markov, Y. (2017, August). The first collision for full SHA-1. In Annual International Cryptology Conference (pp. 570-596). Springer, Cham.

[2] Alani, M. M. (2012, November). Neuro-cryptanalysis of des and triple-DES. In International Conference on Neural Information Processing (pp. 637-646). Springer, Berlin, Heidelberg.

[3] Alallayah, K. M., El-Wahed, W. F., Amin, M., & Alhamami, A. H. (2010). Attack of against simplified data encryption standard cipher system using neural networks. Journal of Computer Science, 6(1), 29.

[4] Akiwate, B., & Desai, V. (2013). Artificial Neural Networks for Cryptanalysis of DES. International Journal of Innovations in Engineering and Technology (IJIET)

[5] Maghrebi, H., Portigliatti, T., & Prouff, E. (2016, December). Breaking cryptographic implementations using deep learning techniques. In International Conference on Security, Privacy, and Applied Cryptography Engineering (pp. 3-26). Springer, Cham.

[6] Lerman, L., Bontempi, G., & Markowitch, O. (2015). A machine learning approach against a masked AES. Journal of Cryptographic Engineering, 5(2), 123-139.

[7] Gilmore, R., Hanley, N., & O'Neill, M. (2015, May). Neural network based attack on a masked implementation of aes. In Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on (pp. 106-111). IEEE.

[8] Heuser, A., & Zohner, M. (2012, May). Intelligent Machine Homicide - Breaking Cryptographic Devices Using Support Vector Machines. In W. Schindler and S. A. Huss, editors, COSADE, volume 7275 of LNCS, pages 249–264. Springer, 2012.

[9] Bartkewitz, T., & Lemke-Rust, K. (2012, November). Efficient template attacks based on probabilistic multi-class support vector machines. In International Conference on Smart Card Research and Advanced Applications (pp. 263-276). Springer, Berlin, Heidelberg.

[10] Hou, S., Zhou, Y., Liu, H., & Zhu, N. (2018). Exploiting Support Vector Machine Algorithm to Break the Secret Key. RADIOENGINEERING, 27(1), 289.

[11] Dileep, A. D., & Sekhar, C. C. (2006, July). Identification of block ciphers using support vector machines. In Neural Networks, 2006. IJCNN'06. International Joint Conference on (pp. 2696-2701). IEEE.