Neural Networking Protocol for Secret Communication

Anas Hasna, Manan Monga,
Michael Korovkin, Taka Shirono, Samyak Jain, Ivan Izhbirdeev

Electrical and Computer Engineering Department
Boston University

ABSTRACT

Secrecy of data and protection against compromise by attackers due to the use of inefficient or easily subvertable security measures employed during data transmission is a huge problem today. However, there are now many encryption schemes, including their respective implementation to address the issue of hiding data. Encryption transforms the plain-text data into random-looking bytes unknown to the attacker. However, in some cases, the very fact that encryption has been used, could lead to compromise of the secrecy of the data. For example, threatening a person to divulge his/her cryptographic keys. Here in this report, we propose a steganographic scheme based on neural networking, being used together to develop a novel data transmission protocol. This report presents a model where cryptography and steganography are implemented together - a message encrypted using the AES-CTR block cipher is hidden in plain sight using our proposed text steganography protocol. We aim to protect users sending and receiving encrypted data by obscuring the fact that such encryption on the data exist and making it look like plain text.

I. Introduction

Modern cryptography has advanced to a level that it is relatively easy to maintain the secrecy of confidential and valuable data with the correct technology and practices. Regardless, it is easy to detect where encryption has been used by statistically analyzing meaningful data and finding that the byte distribution is random. Therefore, cryptography lets us down when we do not want the attackers to know that any information is being hidden at all. This can cause all kinds of problems like censorship by tyrannical governments, oppression of free speech and other techniques knows as "Rubber-hose cryptanalysis".

Steganography is the process of hiding a secret message inside another message in a way that the presence of the secret message is hard to detect for someone from whom we want to keep the message secret. Using steganography, we can transmit sensitive data in disguise as seemingly innocent and understandable data, thereby not raising any suspicion of an attacker intercepting the communication.

However, steganography does not completely abide by the Kerkchhoff's Principle, which states: A cryptosystem should be secure even if everything about the system, except the key, is public knowledge. Standard steganographic schemes mostly use obfuscation as a protection mechanism. The following issue arises, if an enemy knows the process that we are using to obfuscate our messages and can intercept the steganographic packets, they can not only block the covert channel but also read data that was sent. The detection of steganographic packets may not be a trivial problem, but it is still not out of bounds for a patient and resourceful adversary. To make this more robust, we add forward secrecy to our system by creating a hybrid steganographic and cryptographic system which uses the advantages of both approaches. Our proposed steganographic scheme is assumed to abide by Kerkchhoff's principle, as the algorithm is public, and the shared secret is the neural networking model, which is produced by training the model generation algorithm on shared private documents. This report details how we went about creating this system, and in Section II, we talk about the cryptographic methods and other tools used by us in this system and why they are secure. In Section III and we detail our solution and implementation of it. In Section IV, we look at the results of this study. We end this report by concluding how effective our implementation of steganography and cryptography turned out to be.

II. Technology used in this system

We have used and combined various technologies already developed and in use extensively across the industry. The three basic technologies used are:

1. textgenrnn : *training model to generate steganographic texts out of our ciphertexts*
textgenrnn is a Python 3 module build on Tensorflow and Keras frameworks for creating character RNNs (Recurrent Neural Networks). The benefit of texgenrnn over other char-rnn implementations is that it abstracts the process of creating and training such char-rnns to a few lines of code and provides numerous improvements such as character embeddings, attention-weighted averaging and a decaying learning rate. Unlike other internet tutorials on neural networks for text-generation, textgenrnn makes use of deep learning frameworks (as discussed before) to help developers easily train their own neural network models of any size and complexity on any text dataset with a few lines of code on a GPU very quickly (using Google Collaboratory for free). In addition, developers can also use pretrained models to quickly train on a new datasets. Some features of textgenrnn which distinguish it from other text generation models include the capability to work with text at both character-level and word-level, utilization of the CuDNN implementation (in Keras framework) of RNNs when trained on GPUs to get lesser training time as compared to the LSTM implementations (an approximate speedup of 7x), ability to configure RNN size, number of layers, whether to use bidirectional RNNs and lastly, it generates text interactively for customized stories.

2. Advanced Encryption Standard (AES) : *block cipher used for encryption of our message*
We have adhered to the golden rule of encryption, "Never use your own crypto" and used Advanced Encryption Standard (AES), also known as Rijndael, which is a block cipher recommended for usage by The National Institute of Standards and Technology (NIST)[1]. AES can be used in many different modes. AES Counter mode (AES-CTR) is the one used by us for this system. It requires the encryptor to generate a unique per-packet value and communicate this value to the decryptor. This per-packet value is called an initialization vector (IV). The same IV and key combination must not be used more than once otherwise the security of this block cipher is compromised. The encryptor must generate the IV in any manner that ensures uniqueness. Common approaches to IV generation include incrementing a counter for each

packet and linear feedback shift registers (LFSRs). AES-CTR uses the only AES encrypt operation (for both encryption and decryption), making AES-CTR implementations smaller than implementations of many other AES modes. When used properly, AES-CTR mode provides strong confidentiality. Bellare, Desai, Jokipii, Rogaway show in [2] that the privacy guarantees provided by counter mode are at least as strong as those for CBC mode when using the same block cipher.

3. Python Socket Chat Server: *we use this to send and receive our messages*
Our chat server can accept connections from new clients and can identify unique users. It is coded in Python and works well with CLI and is fast and has a good throughput. We have validated that the protocol works on localhost IP. The server uses Python's socket and system library. The messaging model works as follows. First, a chatroom is created by starting a server with a known IP and port. Next, clients can join the chatroom if they know the proper IP and port. When a client joins and sends a message, the message is broadcasted by the server to all clients in the chatroom; each client inputs their secret key upon joining the chatroom, which is used to decrypt all incoming messages and encrypt all outgoing ones. Clients can see their own messages in a decrypted format. Similarly, upon receiving a message, they can see its origin and they can view it in a decrypted format.

III. Solution details and implementation

In this section, we discuss how our solution has been implemented and how a sender Alice can use it to send a secret message to receiver Bob. Also see figure 1 for the dataflow.

1. Alice generates a random key and a neural networking model, then shares for AES-CTR and text generation model and shares it with Bob
2. Alice inputs text into our system
3. The system encrypts the texts and generates a ciphertext with AES-CTR
4. This ciphertext is converted to steganographic text file in the following steps
    (a) Generate 2 suggestions using the model and store them in an array T0

(b) Take the value of the most significant bit = i in ciphertext and use this to start the steganographic file generation as steganotext += T0[i]

(c) This i is used as input into the model, this step will dictate how the generation will go as it generates 2 suggestions and stores them in an array T1

(d) ciphertext = ciphertext[1:] remove the leftmost bit as we are done with it.

(e) This process repeats till there are no more bits left in the ciphertext

5. The steganographic text generated by this process is then sent over socket chat room by Alice

6. Bob receives the steganographic text

7. Uses the text generation model sent by Alice as part of key exchange to generate the 2 initial suggestions and store them in an array T0, this will be the same array as initial sender array

8. Bob then takes the index where the steganotext[0] occurs in T0 and starts generating ciphertext as ciphertext += index

9. Bob also uses the index value as the input for the model, this dictates how the text generation goes

10. This process is repeated for evert steganotext but till the entire steganographic text has been converted to ciphertext.

11. This ciphertext generated is decrypted into the secret message by using the secret key sent during key exchange and AES-CTR

References and Notes

1. We've included in the template file `scifile.tex` a new environment, {`scilastnote`}, that generates a numbered final citation without a corresponding signal in the text. This environment can be used to generate a final numbered reference containing acknowledgments, sources of funding, and the like, per *Science* style.

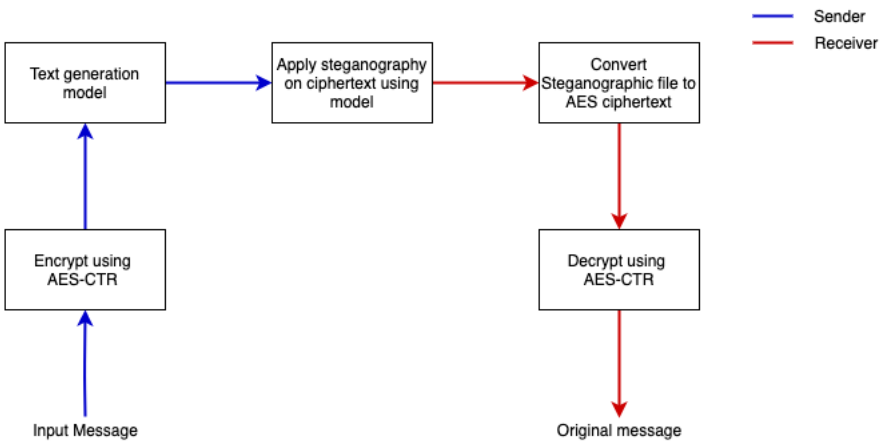

Figure 1: Flow Diagram of presented solution