Crypson

An end-to-end deep learning encryption framework

ECE 519 - Project

Andreas Karatzas

School of Electrical, Computer, and Biomedical Engineering

Embedded Systems Software Lab

Ph.D. Candidate

April 16, 2024

Contents

1. Introduction

With the recent advancements in artificial intelligence, modern neural networks are being deployed to address a wide variety of tasks. One field that has shown interest in applying neural networks is cryptography [1]. Specifically, there are challenges that come with the advancement of computing systems, particularly quantum computing systems, that threaten to break classical encryption techniques. However, neural networks aim to overcome this challenge.

Neural networks are characterized by properties such as a non-linear relationship between input and output, complexity in architecture, and the ability for life-long learning. Cryptography can capitalize on these properties to build frameworks that are near-impossible to crack due to their complexity and successfully address the aforementioned challenges. To that end, this work presents **Crypson**, a framework that leverages modern advances in artificial intelligence to address security challenges in end-to-end encryption. Specifically, *Crypson* is split in 2 parts: ❶ Encoder; and ❷ Decoder.

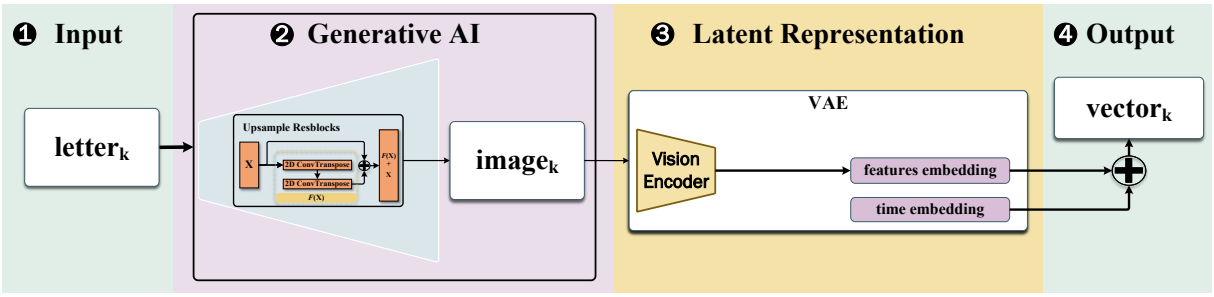


Figure 1.1: Abstract encoder architecture

First, the encoder takes in a sequence of words, i.e., data stream. Then, it tokenizes this sequence of words into letters, formulating a sequence of letters $l_k,\ k \in 0, N$. *Crypson* operates at the letter level, and hence, the framework encrypts and decrypts each letter l_k iteratively until all words and all letters have been successfully decrypted by the receiver. For each letter, the sender initializes a random matrix. This matrix is used as a reference point for the generative model (conditional GAN) [2] to compile an image that is going to represent the letter (inverse OCR task[1]). In this work, I will leverage EMNIST [3] dataset that includes both numerical digits and letters. After generating the image, I will be using a

[1] Inverse optical character recognition (OCR) is the electronic or mechanical conversion of machine-encoded text into images of handwritten text

Variational Auto-Encoder (VAE) [4]. This will transpose the pixel features into a compressed and rich latent form. This is a crucial step in keeping the computational burden of sending the data over the network a cheap operation. The resulting latent vector ("feature embedding" in Figure 1.1) is going to be summed with a time embedding [5], which is a vector of equal length used to mask the latent space, thus making it near-impervious to deciphering it. This creates our encrypted vector k.

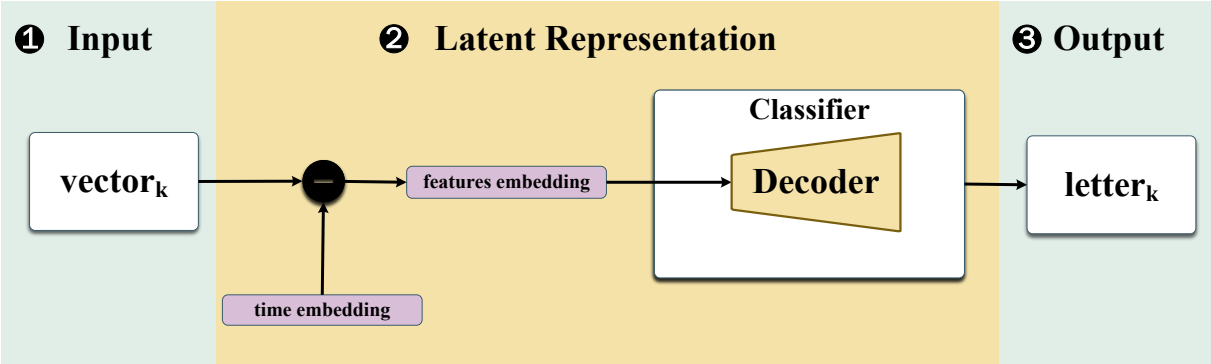


Figure 1.2: Abstract decoder architecture

The decoder is a simpler module. The first step in the decoder is unmasking the latent vector by subtracting the time embedding. This is feasible since the function for generating the time embedding is common between the sender and the receiver. Next, we will be using the decoder module of the trained VAE to transpose the latent features back into a machine-readable letter k.

2. Methodology

Conditional GAN In this research, we employ Conditional Generative Adversarial Networks (CGANs) to generate images conditioned on specific class labels, allowing for more controlled and diverse image synthesis. The CGAN architecture comprises two main components: ❶ a Generator, and ❷ a Discriminator, both of which are crucial for the adversarial learning process.

The **Generator** is designed to map a random noise vector, combined with a class label, to an image. It utilizes an embedding layer to transform the class labels into a continuous vector of the same dimension as the noise vector. These vectors are then element-wise multiplied to produce a conditioned latent vector. This vector is subsequently passed through a series of deconvolutional layers to upscale it to the desired image size. Each deconvolutional layer is accompanied by batch normalization and ReLU activation to stabilize training and encourage the model to learn non-linear transformations effectively. The output layer utilizes the Tanh activation function to scale the pixel values of the generated images to the range [-1, 1], matching the normalization of the training images.

The **Discriminator**'s role is to differentiate between real images from the training dataset and fake images generated by the Generator. It also uses an embedding layer to process the class labels, transforming them into a vector that matches the flattened dimension of the input images. This label embedding is concatenated with the flattened image vector, forming a combined feature vector that serves as input to a series of linear layers. The linear layers progressively downsample the feature vector to a single prediction output. The use of LeakyReLU activation ensures that the model maintains gradient flow during training, which can be critical in preventing the vanishing gradient problem often seen in GANs. The final output layer employs a sigmoid activation to produce a probability indicating the likelihood of the input image being real.

To train our CGAN, we alternately update the Generator and Discriminator using the Adam optimizer. The Generator's goal is to increasingly "fool" the Discriminator by generating images that are indistinguishable from real images, while the Discriminator strives to improve its ability to distinguish between the two sources. The training process involves minimizing respective loss functions for each component. Specifically, binary cross-entropy loss is used for the discriminator to measure its performance in correctly classifying real and generated images and for the generator to maximize the probability of the discriminator making a mistake. This adversarial training stops with the Generator producing high-fidelity images and the Discriminator suitably being adept at classification tasks.

VAE Following the Conditional GAN, we utilize the Variational Autoencoder (VAE) to model the distribution of our data in a latent space, facilitating both generative tasks and efficient data encoding. A VAE consists of two principal components: ❶ an Encoder, and ❷ a Decoder. The Encoder is designed to compress the input data into a condensed representation in the latent space, characterized by two parameters: mean (μ) and variance ($\log \sigma^2$). These parameters define a Gaussian distribution from which we sample to generate latent vectors. This sampling involves reparameterization, which allows the gradient of the loss function to pass through the stochastic sampling process, making it compatible with backpropagation.

The **Encoder** employs convolutional layers, which progressively downsample the input image while increasing the number of feature channels. Each convolution is followed by batch normalization and ReLU activation to maintain numerical stability and introduce non-linear capabilities. The output of these layers is flattened and passed through fully connected layers to produce the μ and $\log \sigma^2$ parameters.

The **Decoder** of the VAE aims to reconstruct the input image from the sampled latent vector. It begins with a fully connected layer that maps the latent vector to a higher-dimensional space and reshapes it to a feature map. This map is then upsampled through a series of transposed convolutional layers, effectively learning to reverse the encoding process. Batch normalization and ReLU activations are again used between layers to help the network learn effectively. The final layer of the Decoder uses a sigmoid activation to ensure that the output pixels are scaled to $[0, 1]$, matching the normalized input values.

The VAE is trained by minimizing a loss function that comprises two terms: the reconstruction loss, which encourages the decoded samples to be as close as possible to the original inputs, and the KL divergence, a regularization term that measures how much the learned latent distribution deviates from the prior distribution (assumed to be a standard Gaussian). This dual-objective loss not only helps in generating high-quality reconstructions but also ensures that the latent space is well-organized and meaningful, providing a robust framework for generating new samples that are variations on the learned data distribution.

Classifier Building on the foundations set by the Variational Autoencoder and Conditional GAN, the classifier is designed to interpret and categorize the latent representations derived from these models. This classifier serves as the deciphering module, decoding messages within the latent space generated by the autoencoder.

The classifier begins with a robust feature extraction process, leveraging a sequence of neural network layers that are initialized with weights from the pre-trained decoder module of the autoencoder. This approach effectively transfers learned features, enhancing the classifier's ability to interpret complex patterns in the data. After cloning the architecture from the VAE's decoder module, we append fully connected layers with an output dimension equal to the number of classes, using a softmax activation to produce a probabilistic distribution over potential class labels. This setup allows the classifier not only to predict the most likely class but also to give insights into the confidence of its predictions across different categories.

3. Experimental Results

In this chapter, I will demonstrate the performance of the conditional GAN, the VAE, and the classifier. To complete the proposed methodology, I also introduce the **Timestep Embedding Generator**, designed to intricately capture temporal patterns within data sequences. The Timestep Embedding Generator can dynamically switch between discrete timestep embeddings and continuous sinusoidal position embeddings, making it adept for both fixed and variable-length sequences. The discrete approach utilizes a standard embedding layer suitable for scenarios with a known and fixed number of timesteps, whereas the continuous method applies a sinusoidal position embedding followed by a multi-layer perceptron (MLP).

3.1 Training and Evaluation of Conditional GAN

The training of the cGAN involved a meticulous setup to balance the adversarial nature of the generator and discriminator. The optimization was performed using the Adam optimizer with a learning rate of 0.0002, a beta value of 0.5 for the generator, and 0.999 for the discriminator. Training was carried out for 200 epochs, ensuring adequate time for the network to stabilize and produce high-quality results. The learning curves for the generator and discriminator losses, depicted in Figure 3.1, demonstrate the convergence of the model over the training epochs, as well as the combined, i.e., sum of the reconstruction and Kullback–Leibler divergence, validation loss.

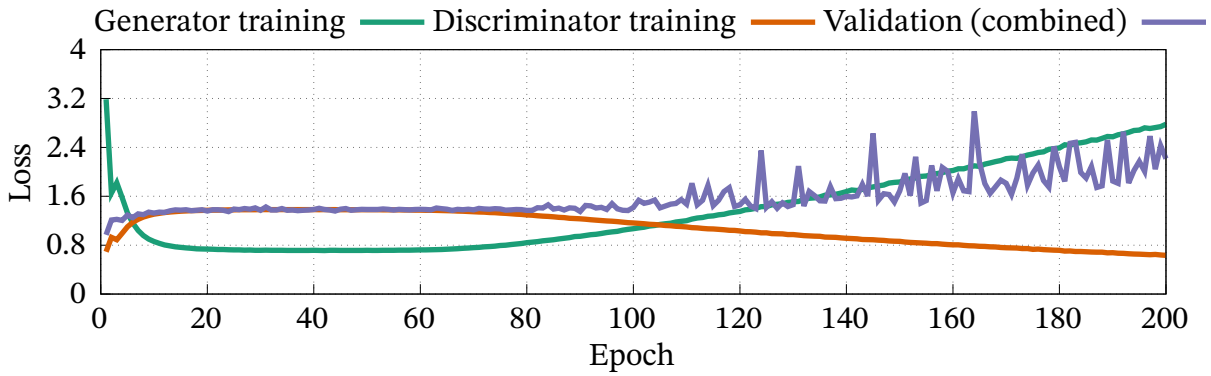


Figure 3.1: Training progress for the cGAN.

3.2 Training and Evaluation of VAE

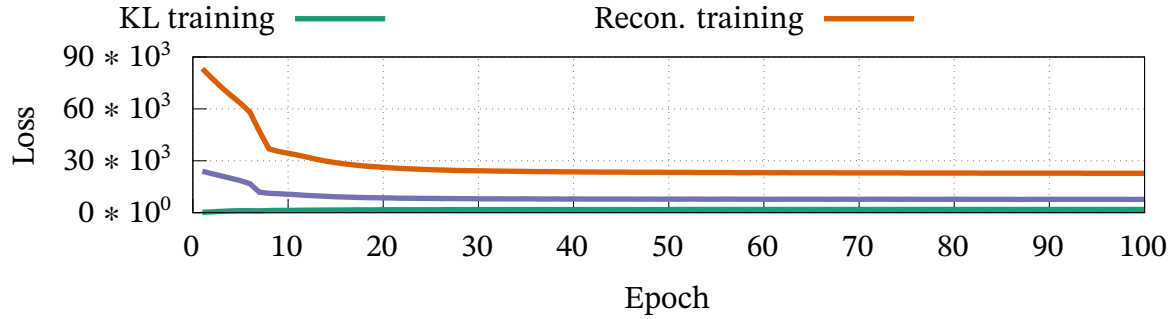
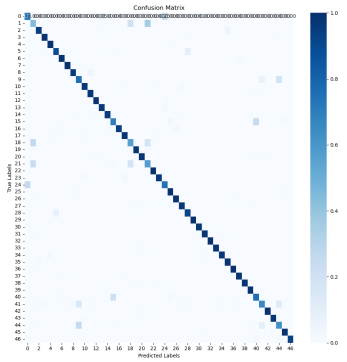


Figure 3.2: Training progress for the VAE.

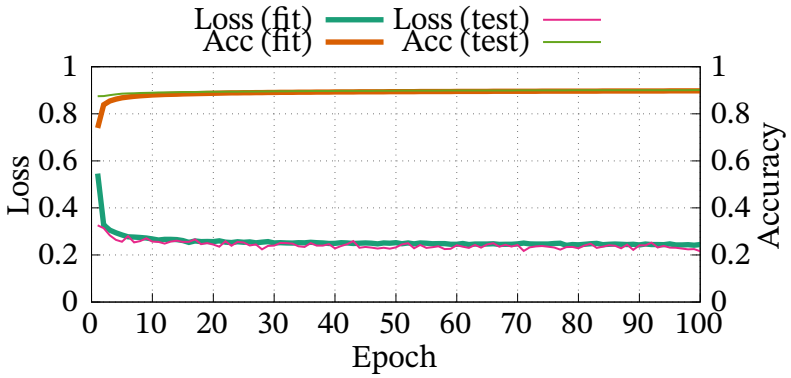
To maximize the trade-off between highly accurate reconstruction of the input data and dimensionality reduction, we set the dimension of the latent space of Crypson's VAE to 8. Training the VAE involved minimizing the reconstruction loss combined with the Kullback-Leibler divergence, a measure of how one probability distribution diverges from a second, expected probability distribution. We used an Adam optimizer with a learning rate of 0.0002 and trained our model for 100 epochs. We also apply a weight of 0.7 for the Kullback-Leibler divergence to reinforce the VAE to compile more accurate and diverse latents. This setup allowed the networks to learn efficient data encodings with sufficient detail retained in the reconstructions, as evidenced by the training and validation loss curves shown in Figure **??**.

3.3 Training and Evaluation of the Classifier

Crypson's classifier is trained using the cross-entropy loss function, which is well-suited for classification tasks. The classifier is trained for 100 epochs, with the weights of the VAE's decoder module kept frozen to maintain the learned representations. Figure 3.3 presents a comprehensive analysis of the classifier's performance. The confusion matrix in Figure 3.3a illustrates the model's ability to accurately classify instances across the predicted and true label space. The strong diagonal pattern suggests that the classifier accurately distinguishes between the different classes. Furthermore, the loss and accuracy curves in Figure 3.3b provide valuable insights into the model's learning progress and generalization ability during training. It is worth noting that certain classes in the dataset exhibit significant visual similarity, posing a challenge for accurate classification. For instance, the lowercase and uppercase letters "f" and "F" share similar structural features, making them difficult to distinguish. Similarly, the characters "s" and "5" have comparable shapes, as do the characters "l", "i", "I", and "1". These similarities can lead to misclassifications, as evidenced by the off-diagonal elements in the confusion matrix. However, it is important to recognize that the current limitations in classification accuracy can be attributed to the inherent constraints of the EMNIST dataset. By utilizing a more comprehensive and representative dataset specifically designed to address the challenges posed by visually similar classes, we anticipate that Crypson's classifier can achieve near-perfect accuracy. This enhanced dataset



(a) Confusion matrix of the trained classifier on the test set.



(b) Loss and accuracy curves for both training and validation phases.

Figure 3.3: Performance evaluation of Crypson's classifier. (a) The confusion matrix demonstrates the model's ability to correctly classify instances across the predicted and true label space. (b) The loss and accuracy curves show the model's learning progress and generalization ability during training.

would enable the classifier to learn more robust and discriminative features, effectively differentiating between even the most visually similar classes.

3.4 Qualitative Evaluation of the Encoder Module

Figures 3.4 and 3.5 depict randomly generated images and reconstructed images by the conditional GAN and VAE, respectively. We observe that the cGAN was able to generate highly realistic sequences that closely mimic the spatial patterns observed in the real data. At the same time, we also observe that the VAE can efficiently compress the input raw pixel space from 32×32 down to just 8 elements, streamlining a rich and highly optimized latent space. The results indicate that the *Encoder* is capable of capturing meaningful latent representations that effectively encrypt the underlying message, demonstrating its potential for practical applications in cybersecurity.

Figure 3.4: Generated samples from the conditional GAN

Figure 3.5: Reconstructed images by the VAE

4. Conclusion

This work will serve as a motivational project to inspire interest in the researched subject. In particular, there are several extensions that can be applied to further advance the innovation behind this work. First, instead of using letters as a target for our cGAN, we can utilize more complex datasets, like ImageNet, to initialize reconfigurable mappings between letter classes and image classes, thus obfuscating the nature of the encoded element even further. Another extension that can be applied is the compression of architecture using modern neural network compression techniques, such as weight pruning and quantization or model distillation. These techniques could yield a computational complexity framework comparable to that of traditional encryption algorithms. Finally, **Crypson** could be extended to support any data stream, such as images and audio, thus assuming a general and impenetrable state-of-the-art encryption framework.

Bibliography

[1] I. Meraouche, S. Dutta, H. Tan, and K. Sakurai, "Neural networks-based cryptography: A survey," *IEEE Access*, vol. 9, pp. 124 727–124 740, 2021.

[2] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[3] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik, "Emnist: Extending mnist to handwritten letters," in *2017 international joint conference on neural networks (IJCNN)*. IEEE, 2017, pp. 2921–2926.

[4] A. Van Den Oord, O. Vinyals *et al.*, "Neural discrete representation learning," *Advances in neural information processing systems*, vol. 30, 2017.

[5] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 10 684–10 695.