

국산암호로 기밀성이 강화된 스테가노그래피에 대한 딥러닝 기반의 스테그아날리시스

<https://youtu.be/gGFVBTxjsBs>

연구 계획

기존 연구들

- 기존의 대부분의 연구들이 임베딩 방식만 구별
- 그러나 이러한 연구들의 데이터는 기밀성이 낮으며 암호화를 진행하지 않았음
- 우리는 기밀성을 향상시키기 위해 임베딩 된 스테가노그래피 이미지에 국산암호 Speck을 사용하여 분석 할 예정
- 우선 32비트인 Speck 암호를 시작으로 성공 시 다양한 국산 암호를 사용해 볼 예정

스테가노그래피 임베딩

- 스테가노그래피에서 가장 유명한 임베딩인 JMOD와 JUNIWARD를 사용

스태가노그래피 / 스테그아날리시스

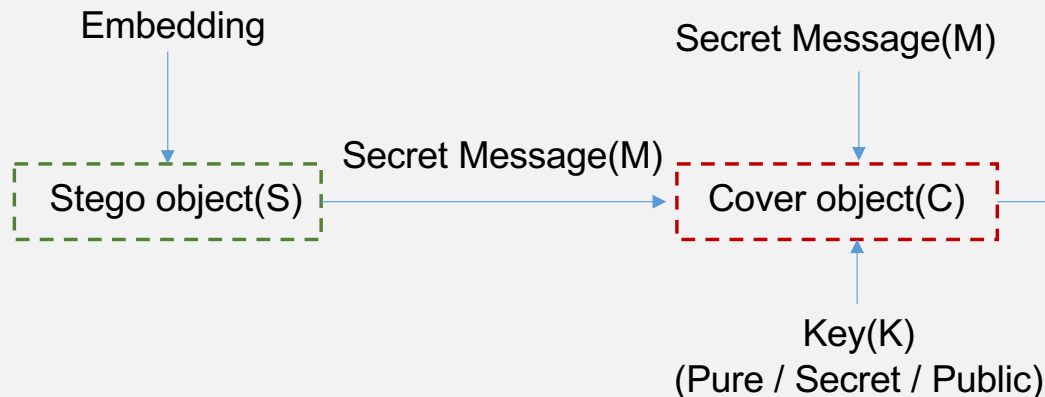
스태가노그래피

- 스태가노그래피는 중요 정보를 숨기기 위해 텍스트, 이미지, 동영상 등 미디어 파일에 데이터를 삽입하는 기술

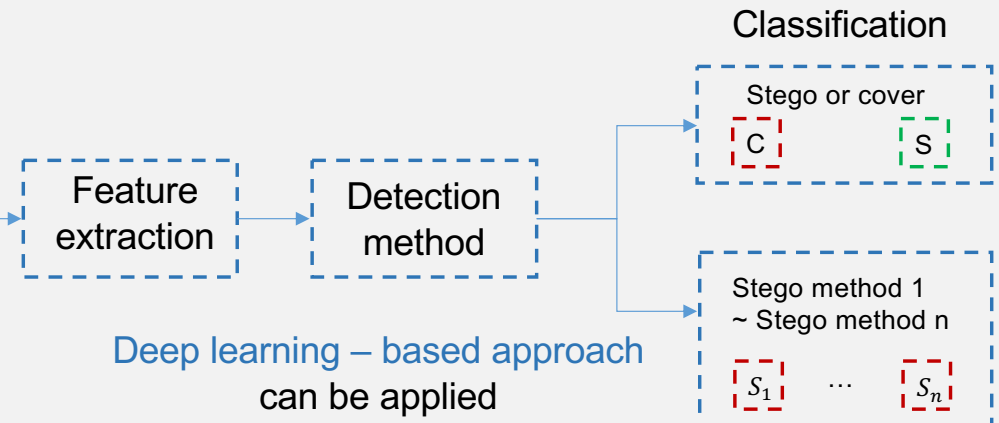
스태그아날리시스

- 스태그아날리시스는 스테가노그래피 기법으로 생성된 스테고 오브젝트를 탐지하는 기술으로 이미지나 동영상 등에서 특징적인 부분을 분석하여 숨겨진 데이터를 찾아내는 역할

Steganography (Embedding)



Steganalysis (Classification)



JMOD, JUNIWARD

스태가노그래피 임베딩

- JMOD, JUNIWARD


내 드라이브 > data1 ▾


유형 ▾


사람 ▾

수정 날짜 ▾


이름 ▾


 S_JUNIWARD

 S_JMOD















```
from google.colab import drive
drive.mount('/content/drive')
```

 Mounted at /content/drive



```
imagePath = '/content/drive/MyDrive/data1'
categories = ["S_JMOD", "S_JUNIWARD"]
```

이름 ▾	
	30000.jpg
	29999.jpg
	29998.jpg
	29997.jpg
	29996.jpg
	29995.jpg
	29994.jpg
	29993.jpg
	29992.jpg
	29991.jpg
	29990.jpg
	29989.jpg
—	

실험 진행

임베딩 된 S_JMOD 이미지 2만8천장을 32x32사이즈로 줄임

```
imagePath = '/Users/gimdeog-yeong/Downloads/'
categories = ["S_JMOD"]
nb_classes = len(categories)
```

```
image_w = 32
image_h = 32
```

```
X = []
Y = []
```

```
count = 0
count1 = 0
count2 = 0
```

```
1700 1 1
1800 1 1
1900 1 1
2000 1 1
2100 1 1
2200 1 1
2300 1 1
2400 1 1
...
28100 1 1
28126
28126
the number of data : 28126
```

```
image_dir = imagePath+'/'+cate+'/'

for top, dir, f in os.walk(image_dir):
    count2 += 1
    print(f)
    for filename in f:
        #print(image_dir+filename)
        img = cv2.imread(image_dir+filename)
        img = cv2.resize(img, None, fx=image_w/img.shape[1], fy=image_h/img.shape[0])
        X.append(img/256)
        Y.append(idx) # label
        count += 1
        if(count % 100 == 0):
            print(count, count1, count2)
```

```
print(len(X))
print(len(Y))
```

```
X_np = np.array(X) #data
Y_np = np.array(Y) #label
```

```
print("the number of data :", len(X_np))
```

✓ 55.9s

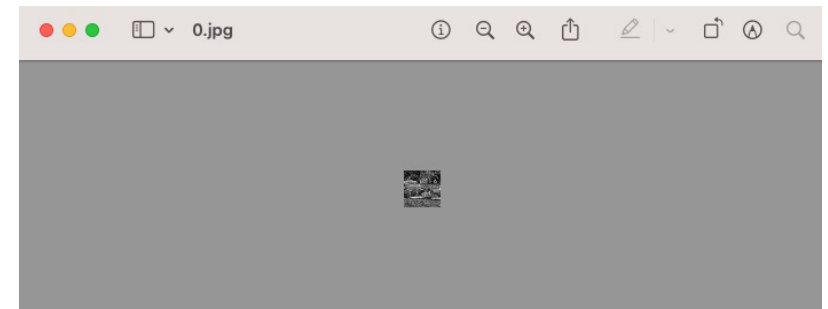
실험 진행

이미지가 3차원 배열이고 컬러로 진행하게 되면 계산이 복잡하여 흑백 컬러로 변환 (사이즈는 32x32)

```
from glob import glob

X_np_uint8 = (X_np * 255).astype(np.uint16)
for i in range(28126):
    b,g,r = cv2.split(X_np_uint8[i])
    image222_gray=((b+g+r)/3).astype(np.uint8)
    cv2.imwrite('/Users/gimdeog-yeong/Downloads/gray1/S_JMOD1/%d.jpg' %i, image222_gray)
```

✓ 2.8s



S_JMOD1			
이름	수정일	크기	종류
0.jpg	오늘 오후 12:57	1KB	JPEG 이미지
1.jpg	오늘 오후 12:57	1KB	JPEG 이미지
2.jpg	오늘 오후 12:57	1KB	JPEG 이미지
3.jpg	오늘 오후 12:57	894바이트	JPEG 이미지
4.jpg	오늘 오후 12:57	1KB	JPEG 이미지
5.jpg	오늘 오후 12:57	1KB	JPEG 이미지
6.jpg	오늘 오후 12:57	976바이트	JPEG 이미지
7.jpg	오늘 오후 12:57	768바이트	JPEG 이미지
8.jpg	오늘 오후 12:57	961바이트	JPEG 이미지
9.jpg	오늘 오후 12:57	977바이트	JPEG 이미지
10.jpg	오늘 오후 12:57	808바이트	JPEG 이미지
11.jpg	오늘 오후 12:57	776바이트	JPEG 이미지
12.jpg	오늘 오후 12:57	1KB	JPEG 이미지
13.jpg	오늘 오후 12:57	1KB	JPEG 이미지
14.jpg	오늘 오후 12:57	1KB	JPEG 이미지
15.jpg	오늘 오후 12:57	921바이트	JPEG 이미지
16.jpg	오늘 오후 12:57	848바이트	JPEG 이미지

실험 진행

S_JMOD 이미지 구조를 3차원 배열로 변경

```
from PIL import Image

imagePath = '/Users/gimdeog-yeong/Downloads/gray2/S_JMOD1/'
#categories = ["S_JMOD1"]

#nb_classes = len(categories)

X1 = []
Y1 = []

img_list = os.listdir(imagePath) #디렉토리 내 모든 파일 불러오기

image_dir = imagePath
```

```
img_list_np = []

for i in img_list:
    img = Image.open(image_dir + i)
    img_array = np.array(img)
    img_list_np.append(img_array)
    print(i, " 추가 완료 - 구조:", img_array.shape) # 불러온 이미지의 차원 확인 (세로X가로X색)
    #print(img_array.T.shape) #축변경 (색X가로X세로)

img_np = np.array(img_list_np) #리스트를 numpy로 변환
print(img_np.shape)
```

✓ 5.9s

```
...
6419.jpg 추가 완료 - 구조: (32, 32)
1376.jpg 추가 완료 - 구조: (32, 32)
13631.jpg 추가 완료 - 구조: (32, 32)
(28126, 32, 32)
```

Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings...](#)

+ 코드

+ Markdown

실험 진행

국산 암호 Speck으로 암호화 진행 후 npy 파일 저장 데이터 / 라벨로 나누어 저장

```
num_img = len(img_np)
rows = 32

row = []

cp = [[[0 for i in range(32)] for j in range(32)] for k in range(28126)]

for i in range(28126):
    for j in range(32):
        for k in range(32):
            cipher = SpeckCipher(0x1f1e1d1c1b1a1918, 64, 32, 'ECB')
            cp[i][j][k] = cipher.encrypt(img_np[i][j][k])
```

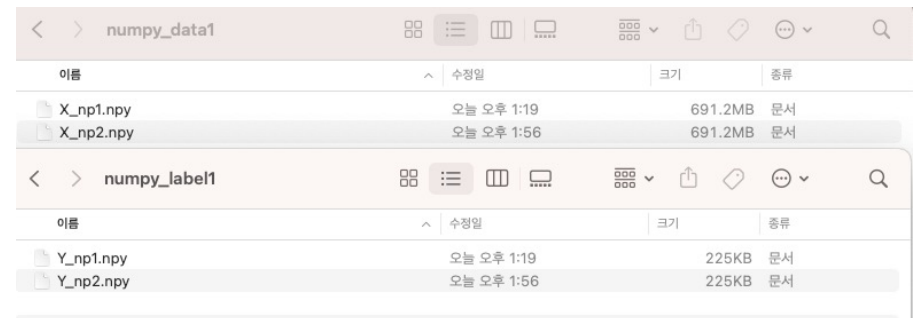
✓ 12m 28.0s

```
# save
np.save('/Users/gimdeog-yeong/Downloads/stg_data1/numpy_data1/X_np1', X_np)
np.save('/Users/gimdeog-yeong/Downloads/stg_data1/numpy_label1/Y_np1', Y_np)
```

✓ 2.4s



이름	수정일	크기	종류
X_np1.npy	오늘 오후 1:19	691.2MB	문서



이름	수정일	크기	종류
Y_np1.npy	오늘 오후 1:19	225KB	문서

실험 진행

코랩에 로드하여 arr, B이라는 변수에 데이터를 저장

```
[ ] #load
X_stg = np.load('/content/drive/MyDrive/stg_data/numpy_data/X_np.npy', allow_pickle=True)
Y_stg = np.load('/content/drive/MyDrive/stg_data/numpy_label/Y_np.npy', allow_pickle=True)

[ ] #load
X_stg_U = np.load('/content/drive/MyDrive/stg_data/numpy_data/X_np_U.npy', allow_pickle=True)
Y_stg_U = np.load('/content/drive/MyDrive/stg_data/numpy_label/Y_np_U.npy', allow_pickle=True)
```

```
[ ] arr = np.empty((56252, 32, 32), dtype=np.int64)
```

```
[ ] for i in range(28126):
    arr[i] = X_stg[i]
    for i in range(28126):
        arr[i+28126] = X_stg_U[i]
```

```
[ ] arr.shape
```

```
(56252, 32, 32)
```

```
[ ] B = np.append(Y_stg, Y_stg_U)
```

더블클릭 또는 Enter 키를 눌러 수정

B.shape

```
(56252, )
```

실험 진행

x_train, y_train에 arr과 B를 넣어 실험 진행

```
[ ] x_train, y_train, x_val, y_val = train_test_split(arr,B,test_size=0.2,train_size=0.8)
    #x_train, y_train, x_test, y_test = train_test_split(x_train,y_train,test_size=0.1,train_size=0.9)
```

```
inp = tf.keras.layers.Input(shape=(x_train.shape[1:]))
print('check')
print(inp)
print(x_train.shape)

conv1 = tf.keras.layers.Conv1D(256, kernel_size=16, strides= 1)(inp)
conv1 = tf.keras.layers.BatchNormalization()(conv1)
conv1 = tf.keras.layers.Activation('relu')(conv1)

conv1 = tf.keras.layers.Conv1D(64, kernel_size=8, strides = 1)(conv1)
conv1 = tf.keras.layers.BatchNormalization()(conv1)
conv1 = tf.keras.layers.Activation('relu')(conv1)

conv1 = tf.keras.layers.Conv1D(32, kernel_size=4, strides = 1)(conv1)
conv1 = tf.keras.layers.BatchNormalization()(conv1)
conv1 = tf.keras.layers.Activation('relu')(conv1)

conv1 = tf.keras.layers.Flatten()(conv1)
out = tf.keras.layers.Dense(1, activation='sigmoid')(conv1)

model = tf.keras.Model(inputs=inp, outputs=out)

model.summary()

s = 10 * len(x_train) // 32

lr = keras.optimizers.schedules.ExponentialDecay(0.0001, s, 0.001)
opt = keras.optimizers.Adam(lr)

model.compile(optimizer=opt,
              loss='binary_crossentropy', # categorical_crossentropy
              metrics=['accuracy'])
```

실험 진행

실험 결과

```
model = model_9
```

Layer (type)	Output Shape	Param #
input_10 (InputLayer)	[(None, 32, 32)]	0
conv1d_27 (Conv1D)	(None, 17, 256)	131328
batch_normalization_27 (Batch Normalization)	(None, 17, 256)	1024
activation_27 (Activation)	(None, 17, 256)	0
conv1d_28 (Conv1D)	(None, 10, 64)	131136
batch_normalization_28 (Batch Normalization)	(None, 10, 64)	256
activation_28 (Activation)	(None, 10, 64)	0
conv1d_29 (Conv1D)	(None, 7, 32)	8224
batch_normalization_29 (Batch Normalization)	(None, 7, 32)	128
activation_29 (Activation)	(None, 7, 32)	0
flatten_9 (Flatten)	(None, 224)	0
dense_9 (Dense)	(None, 1)	225

```
=====
```

Total params: 272321 (1.04 MB)
Trainable params: 271617 (1.04 MB)
Non-trainable params: 704 (2.75 KB)

```
] model.fit(x_train, x_val, epochs=10, batch_size=32, validation_data=(y_train, y_val), verbose=2)
```

```
] Epoch 1/10
1407/1407 - 66s - loss: 0.0011 - accuracy: 1.0000 - val_loss: 9.7600e-04 - val_accuracy: 1.0000 - 66s/epoch - 47ms/step
Epoch 2/10
1407/1407 - 76s - loss: 4.7325e-04 - accuracy: 1.0000 - val_loss: 4.4084e-04 - val_accuracy: 1.0000 - 76s/epoch - 54ms/step
Epoch 3/10
1407/1407 - 58s - loss: 2.1293e-04 - accuracy: 1.0000 - val_loss: 2.0764e-04 - val_accuracy: 1.0000 - 58s/epoch - 41ms/step
Epoch 4/10
1407/1407 - 45s - loss: 9.9893e-05 - accuracy: 1.0000 - val_loss: 9.8326e-05 - val_accuracy: 1.0000 - 45s/epoch - 32ms/step
Epoch 5/10
1407/1407 - 46s - loss: 4.7909e-05 - accuracy: 1.0000 - val_loss: 4.9776e-05 - val_accuracy: 1.0000 - 46s/epoch - 32ms/step
Epoch 6/10
1407/1407 - 70s - loss: 2.3264e-05 - accuracy: 1.0000 - val_loss: 2.5050e-05 - val_accuracy: 1.0000 - 70s/epoch - 50ms/step
Epoch 7/10
1407/1407 - 76s - loss: 1.1396e-05 - accuracy: 1.0000 - val_loss: 1.2976e-05 - val_accuracy: 1.0000 - 76s/epoch - 54ms/step
Epoch 8/10
1407/1407 - 70s - loss: 5.6235e-06 - accuracy: 1.0000 - val_loss: 6.4987e-06 - val_accuracy: 1.0000 - 70s/epoch - 50ms/step
Epoch 9/10
1407/1407 - 72s - loss: 2.7943e-06 - accuracy: 1.0000 - val_loss: 3.3094e-06 - val_accuracy: 1.0000 - 72s/epoch - 51ms/step
Epoch 10/10
1407/1407 - 54s - loss: 1.3947e-06 - accuracy: 1.0000 - val_loss: 1.7614e-06 - val_accuracy: 1.0000 - 54s/epoch - 38ms/step
<keras.src.callbacks.History at 0x7ba8fef86a40>
```

감사합니다.