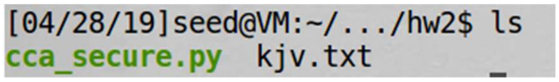
Eduard Klimenko
Anthony Trang
Ryan Moe

Authenticated Encryption

For this encryption and authentication program, we opted to use AES-256 in CTR mode, and HMAC-SHA-256. As a result, we require a key and IV for encryption, as well as a key for authentication. We chose this scheme because we know AES-256-CTR to be CPA secure. When conducted properly, it is confidential on its own, short of a CCA attack. HMAC-SHA-256 is a secure MAC, and so provides authentication in its own right. When added to the ciphertext, it makes the result CCA secure. This is because CCA attacks rely on altering the ciphertext before feeding the result to a decryption oracle, but an altered ciphertext would invalidate the authentication, causing the oracle to reject the message. In other words, by authenticating we improve the confidentiality, and also ensure integrity by detecting tampering. This is only achievable by using the encrypt-then-authenticate approach.

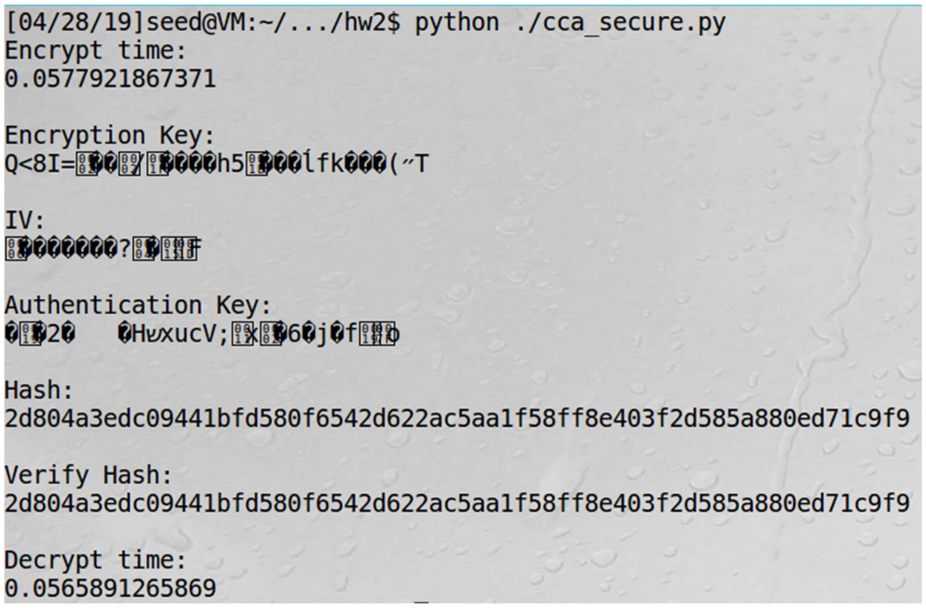
In one trial, the program took approximately 0.05779 seconds to encrypt the bible. The encrypted file was the same size as the plaintext, plus the 80 bytes required for the IV and MAC.

Pre-condition:
Python 2.7
kjv.txt in directory



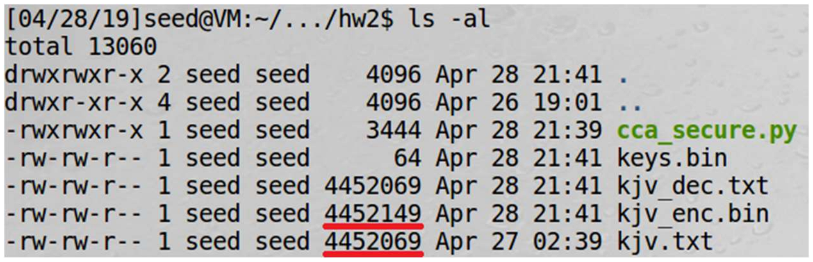
```
[04/28/19]seed@VM:~/.../hw2$ ls
cca_secure.py  kjv.txt
```

We observe that encrypting a 4.4MB file takes just 57.8 milliseconds.



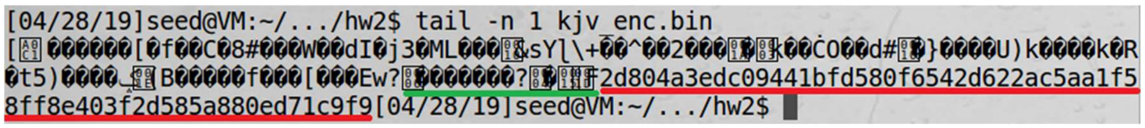
```
[04/28/19]seed@VM:~/.../hw2$ python ./cca_secure.py
Encrypt time:
0.0577921867371

Encryption Key:
Q<8I=�����/���h5����lfk���(~T

IV:
��������?����F

Authentication Key:
���2�   �HuxucV;�x��6�j�f���

Hash:
2d804a3edc09441bfd580f6542d622ac5aa1f58ff8e403f2d585a880ed71c9f9

Verify Hash:
2d804a3edc09441bfd580f6542d622ac5aa1f58ff8e403f2d585a880ed71c9f9

Decrypt time:
0.0565891265869
```

The encrypted file is the 80 bytes larger than the original. The IV (16 bytes) is appended to the end of the file as well as the HMAC (64 bytes). Since we are using CTR mode, we encrypt byte by byte and there is no need for padding unlike CBC mode. This simplifies the implementation.



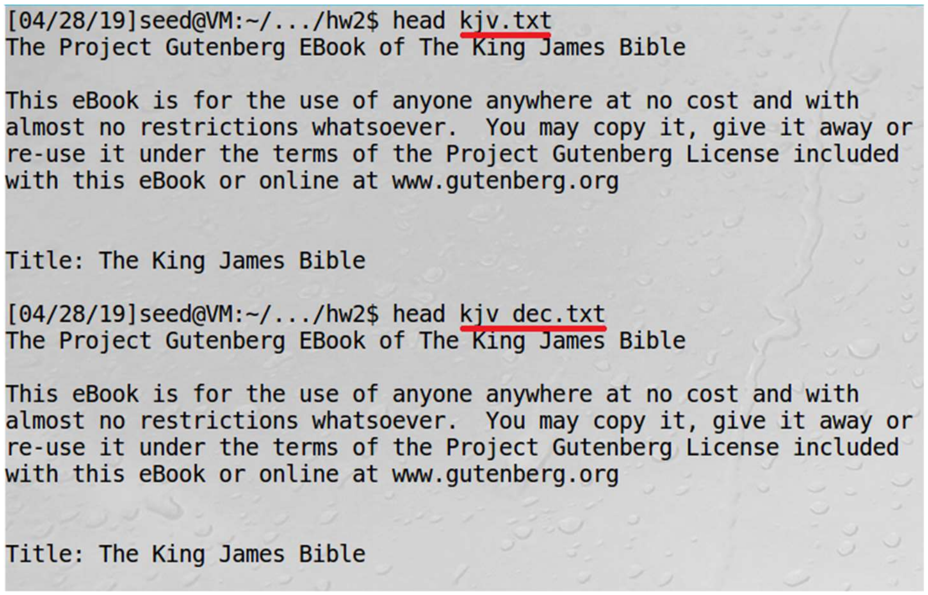
```
[04/28/19]seed@VM:~/.../hw2$ ls -al
total 13060
drwxrwxr-x 2 seed seed    4096 Apr 28 21:41 .
drwxr-xr-x 4 seed seed    4096 Apr 26 19:01 ..
-rwxrwxr-x 1 seed seed    3444 Apr 28 21:39 cca_secure.py
-rw-rw-r-- 1 seed seed      64 Apr 28 21:41 keys.bin
-rw-rw-r-- 1 seed seed 4452069 Apr 28 21:41 kjv_dec.txt
-rw-rw-r-- 1 seed seed 4452149 Apr 28 21:41 kjv_enc.bin
-rw-rw-r-- 1 seed seed 4452069 Apr 27 02:39 kjv.txt
```

Looking at the end of the encrypted file we can see IV (underlined in green) and the HMAC (underlined in red).



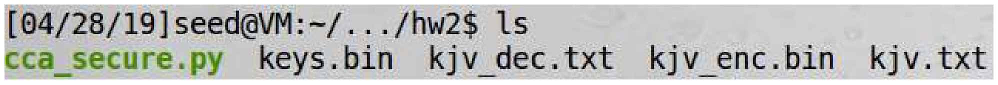
```
[04/28/19]seed@VM:~/.../hw2$ tail -n 1 kjv_enc.bin
[��������[�f��C�8#���W��dI�j3�ML����&sY⌊\+��^��2��������k��ĊO��d#���}����U)k����k�R
�t5)����_��B�����f���[���Ew?���������?�����2d804a3edc09441bfd580f6542d622ac5aa1f5
8ff8e403f2d585a880ed71c9f9[04/28/19]seed@VM:~/.../hw2$
```

Verify decrypted text.



```
[04/28/19]seed@VM:~/.../hw2$ head kjv.txt
The Project Gutenberg EBook of The King James Bible

This eBook is for the use of anyone anywhere at no cost and with
almost no restrictions whatsoever.  You may copy it, give it away or
re-use it under the terms of the Project Gutenberg License included
with this eBook or online at www.gutenberg.org


Title: The King James Bible

[04/28/19]seed@VM:~/.../hw2$ head kjv_dec.txt
The Project Gutenberg EBook of The King James Bible

This eBook is for the use of anyone anywhere at no cost and with
almost no restrictions whatsoever.  You may copy it, give it away or
re-use it under the terms of the Project Gutenberg License included
with this eBook or online at www.gutenberg.org


Title: The King James Bible
```

Post-condition:
Keys used for both encryption and authentication, encrypted file, decrypted file



```
[04/28/19]seed@VM:~/.../hw2$ ls
cca_secure.py  keys.bin  kjv_dec.txt  kjv_enc.bin  kjv.txt
```

cca_secure.py:

```
# Eduard Klimenko
# Uses aes-256-ctr and hmac-sha-256 to implement cca secure communication

from Crypto.Cipher import AES
from Crypto.Util import Counter
from Crypto import Random
import binascii
import hmac
import hashlib
import base64
import time

# Encryption and Decryption methods obtained from the web.
# AES supports multiple key sizes: 16 (AES128), 24 (AES192), or 32 (AES256).
key_bytes = 32

# Takes as input a 32-byte key and an arbitrary-length plaintext and returns a
# pair (iv, ciphtertext). "iv" stands for initialization vector.
def encrypt(key, plaintext):
	assert len(key) == key_bytes

	# Choose a random, 16-byte IV.
	iv = Random.new().read(AES.block_size)

	# Convert the IV to a Python integer.
	iv_int = int(binascii.hexlify(iv), 16)

	# Create a new Counter object with IV = iv_int.
	ctr = Counter.new(AES.block_size * 8, initial_value=iv_int)

	# Create AES-CTR cipher.
	aes = AES.new(key, AES.MODE_CTR, counter=ctr)

	# Encrypt and return IV and ciphertext.
	ciphertext = aes.encrypt(plaintext)
	return (iv, ciphertext)

# Takes as input a 32-byte key, a 16-byte IV, and a ciphertext, and outputs the
# corresponding plaintext.
def decrypt(key, iv, ciphertext):
	assert len(key) == key_bytes

	# Initialize counter for decryption. iv should be the same as the output of
	# encrypt().
	iv_int = int(iv.encode('hex'), 16)
	ctr = Counter.new(AES.block_size * 8, initial_value=iv_int)

	# Create AES-CTR cipher.
	aes = AES.new(key, AES.MODE_CTR, counter=ctr)

	# Decrypt and return the plaintext.
	plaintext = aes.decrypt(ciphertext)
	return plaintext

# Generates a new key for AES
key1 = Random.new().read(key_bytes)

# reads input file
fin=open("/home/seed/Desktop/hw2/kjv.txt","r")
fin_data=fin.read()
fin.close()

# measures time it takes to encrypt
start = time.time()
```

```
(iv, ciphertext) = encrypt(key1, fin_data)
end = time.time()
print("Encrypt time: ")
print(end - start)
print("\nEncryption Key:")
print(key1)
print("\nIV:")
print(iv)

# Generates a new key for HMAC
key2 = Random.new().read(key_bytes)
print("\nAuthentication Key:")
print(key2)

# calculates MAC(key2,message)
h = hmac.new( key2, ciphertext, hashlib.sha256 )
print("\nHash:")
print( h.hexdigest() )

# writes ENC(m), iv, HMAC(m) to a new file
fout = open("/home/seed/Desktop/hw2/kjv_enc.bin", "w")
fout.write(ciphertext)
fout.write(iv)
fout.write(h.hexdigest())
fout.close()

# writes generated keys to file
# encryption key followed by the authentication key
fout1 = open("/home/seed/Desktop/hw2/keys.bin", "w")
fout1.write(key1)
fout1.write(key2)
fout1.close()

# reads encrypted file that was just generated
fin2=open("/home/seed/Desktop/hw2/kjv_enc.bin","rb")
fin2_data=fin2.read()
fin2.close()

# calculates the hash to check for authenticity
# ingore last 80 bytes
h2 = hmac.new( key2, (fin2_data[:-80]), hashlib.sha256 )
print("\nVerify Hash:")
print( h2.hexdigest() )

# print("\nDo they match?:")
# print h.hexdigest() == h2.hexdigest() # if true then it is authentic!

# grabs the appended IV and Hash and times decryption
start2 = time.time()
plaintext = decrypt(key1, (fin2_data[-80:-64]), (fin2_data[:-80]))
end2 = time.time()
print("\nDecrypt time: ")
print(end2 - start2)

# writes decrypted text to new file
fout2 = open("/home/seed/Desktop/hw2/kjv_dec.txt", "w")
fout2.write(plaintext)
fout2.close()
```