GCM Mode

Attack on ciphertext's integrity

- Attacker makes changes to ciphertext (Line 2)

```
data = b'The quick brown fox jumps over the lazy dog'

# Encrypt the entire data
cipher = AES.new(key, AES.MODE_OFB, iv)
ciphertext  = bytearray(cipher.encrypt(data))          ①

# Change the 10th byte of the ciphertext
ciphertext[10] = 0xE9                                  ②

# Decrypt the ciphertext
cipher = AES.new(key, AES.MODE_OFB, iv)
plaintext = cipher.decrypt(ciphertext)                 ③
print("Original  Plaintext: {0}".format(data))
print("Decrypted Plaintext: {0}".format(plaintext))
```

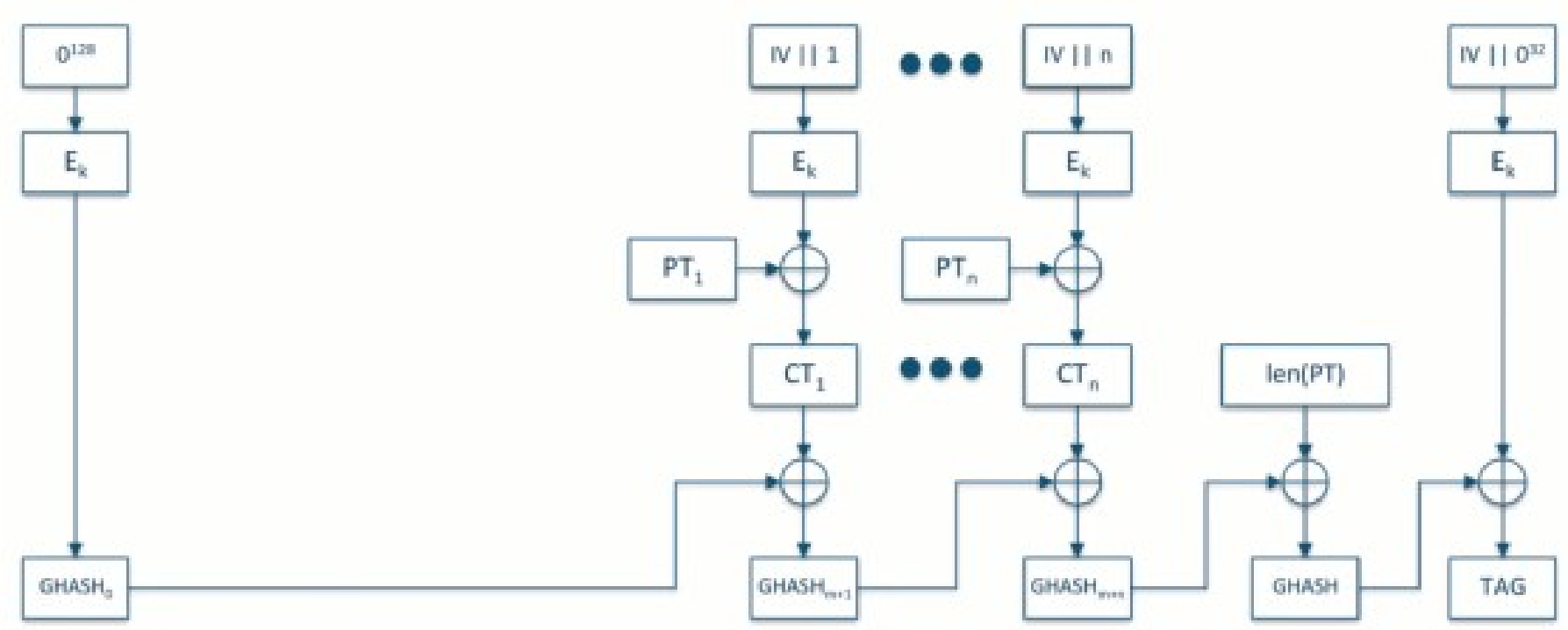
- Result

```
Original  Plaintext: b'The quick brown fox jumps over the lazy dog'
Decrypted Plaintext: b'The quick grown fox jumps over the lazy dog'
```

Authenticated Encryption

- To protect the integrity, the sender needs to generate a Message Authentication Code (MAC) from the ciphertext using a secret shared by the sender and the receiver.
- The MAC and the ciphertext will be sent to the receiver, who will compute a MAC from the received ciphertext.
- If the MAC is the same as the one received, the ciphertext is not modified.
- Two operations are needed to achieve integrity of ciphertext: one for encrypting data and other for generating MAC.
- **Authenticated encryption** combines these two separate operations into one encryption mode. E.g GCM, CCM, OCB

The GCM Mode


0^{128}
E_k
IV || 1
E_k
PT_1
CT_1
IV || n
E_k
PT_n
CT_n
len(PT)
IV || 0^{32}
E_k
$GHASH_0$
$GHASH_{m+1}$
$GHASH_{m+n}$
GHASH
TAG

Programming using the GCM Mode

```
#!/usr/bin/python3

from Crypto.Cipher import AES
from Crypto.Util import Padding

key_hex_string = '00112233445566778899AABBCCDDEEFF'
iv_hex_string  = '000102030405060708090A0B0C0D0E0F'
key = bytes.fromhex(key_hex_string)
iv  = bytes.fromhex(iv_hex_string)
data = b'The quick brown fox jumps over the lazy dog'

# Encrypt the data
cipher = AES.new(key, AES.MODE_GCM, iv)                    ①
cipher.update(b'header')                                   ②
ciphertext  = bytearray(cipher.encrypt(data))
print("Ciphertext: {0}".format(ciphertext.hex()))

# Get the MAC tag
tag = cipher.digest()                                      ③
print("Tag: {0}".format(tag.hex()))
```

The unique part of the above code is the tag generation and verification.
In Line 3 , we use the **digest()** to get the authentication tag, which is generated from the ciphertext.

Programming using the GCM Mode

```
# Corrupt the ciphertext
ciphertext[10] = 0x00                                    ④

# Decrypt the ciphertext
cipher = AES.new(key, AES.MODE_GCM, iv)
cipher.update(b'header')                                 ⑤
plaintext = cipher.decrypt(ciphertext)
print("Plaintext: {0}".format(plaintext))

# Verify the MAC tag
try:
   cipher.verify(tag)                                    ⑥
except:
   print("*** Authentication failed ***")
else:
   print("*** Authentication is successful ***")
```

In Line 6 , after feeding the ciphertext to the cipher, we invoke **verify()** to verify whether the tag is still valid.

Experiment - GCM Mode

- We modify the ciphertext by changing the 10th byte to (0x00)
- Decrypt the modified ciphertext and verify tag

```
$ enc_gcm.py
Ciphertext: ed1759cf244fa97f87de552c1...a11d
Tag: 701f3c84e2da10aae4b76c89e9ea8427
Plaintext: b'The quick 7rown fox jumps over the lazy dog'
*** Authentication failed ***
```