Programming using Crypto APIs

- Example of encryption & decryption using PyCryptodome
- Experiment: Attacking the integrity of ciphertext
- Authenticated Encryption (AE) with the GCM mode of operation

Programming using Crypto APIs

```
#!/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'
print("Length of data: {0:d}”.format(len(data))) # 43 bytes
```

- Here we use PyCryptodome package's APIs
- Setup:
  - Set the Key & IV (hex string)
  - Define the data (byte literal)

Programming using Crypto APIs

```
# Encrypt the data piece by piece
cipher = AES.new(key, AES.MODE_CBC, iv)
ciphertext  = cipher.encrypt(data[0:32])
ciphertext += cipher.encrypt(Padding.pad(data[32:], 16))
print("Ciphertext: {0}".format(ciphertext.hex()))

# Encrypt the entire data
cipher = AES.new(key, AES.MODE_CBC, iv)
ciphertext = cipher.encrypt(Padding.pad(data, 16))
print("Ciphertext: {0}".format(ciphertext.hex()))

# Decrypt the ciphertext
cipher = AES.new(key, AES.MODE_CBC, iv)
plaintext = cipher.decrypt(ciphertext)
print("Plaintext: {0}".format(Padding.unpad(plaintext, 16)))
```

Approach 1:

1. Setup
2. Initialize cipher
3. Encrypts first 32 bytes of data
4. Encrypts the rest of the data
5. Initialize cipher (start new chain)
6. Encrypt the entire data
7. Initialize cipher for decryption
8. Decrypt

Programming using Crypto APIs

```
# Encrypt the data piece by piece
cipher = AES.new(key, AES.MODE_CBC, iv)
ciphertext  = cipher.encrypt(data[0:32])
ciphertext += cipher.encrypt(Padding.pad(data[32:], 16))
print("Ciphertext: {0}".format(ciphertext.hex()))

# Encrypt the entire data
cipher = AES.new(key, AES.MODE_CBC, iv)
ciphertext = cipher.encrypt(Padding.pad(data, 16))
print("Ciphertext: {0}".format(ciphertext.hex()))

# Decrypt the ciphertext
cipher = AES.new(key, AES.MODE_CBC, iv)
plaintext = cipher.decrypt(ciphertext)
print("Plaintext: {0}".format(Padding.unpad(plaintext, 16)))
```

Approach 2:

1. Setup
2. Initialize cipher
3. Encrypts first 32 bytes of data
4. Encrypts the rest of the data
5. Initialize cipher (start new chain)
6. Encrypt the entire data
7. Initialize cipher for decryption
8. Decrypt

Programming using Crypto APIs

- Modes that do not need padding include: CFB, OFB, and CTR.

```
# Encrypt the data piece by piece
cipher = AES.new(key, AES.MODE_OFB, iv)
ciphertext  = cipher.encrypt(data[0:20])
ciphertext += cipher.encrypt(data[20:])
```

- For these modes, the data fed into the **encrypt()** method can have an arbitrary length, and no padding is needed (everything else is the same)

Revisiting <u>data integrity</u> now that we have new tools…

Attacks on Ciphertext Integrity

- Suppose an attacker makes changes to the ciphertext:

```
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))

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

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

- Result:

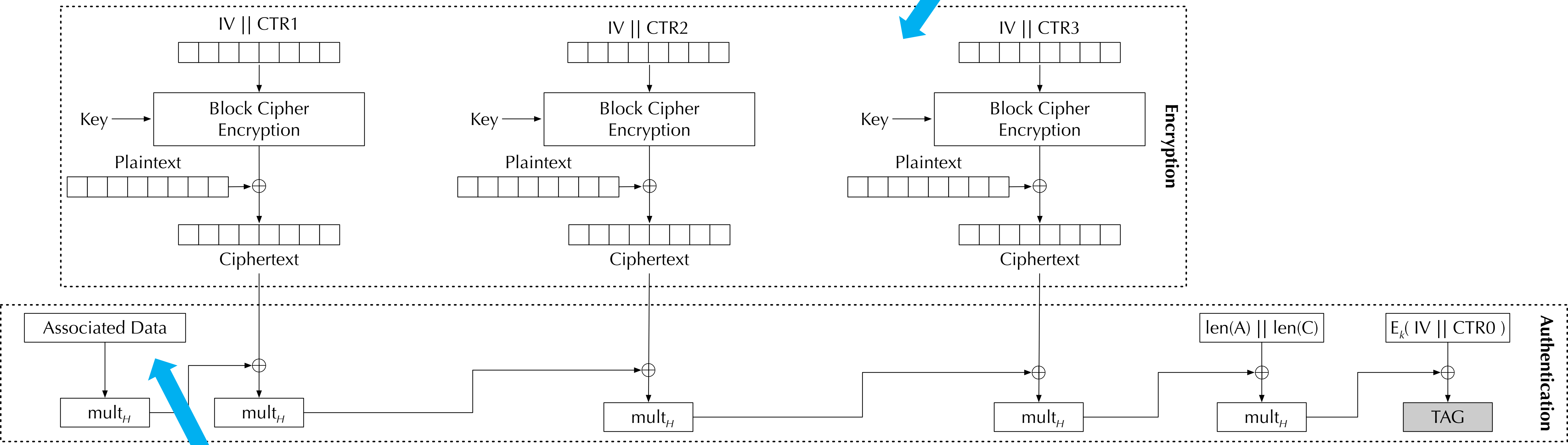
```
 Original Plaintext: b'The quick brown fox jumps over the lazy dog'
Decrypted Plaintext: b'The quick grown fox jumps over the lazy dog'
```

Authenticated Encryption (AE)

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

The Galois/Counter Mode (GCM)

Encrypt data using a familiar block cipher & mode of operation
(AES w/ CTR mode)



Associated Data:
unencrypted, but integrity protected
(e.g., packet headers)

MAC ("Tag")
that covers integrity of ciphertext and associated data

Programming using the GCM Mode

```
#!/usr/bin/python3

# ...snip...
data = b'The quick brown fox jumps over the lazy dog'

# Encrypt the data
cipher = AES.new(key, AES.MODE_GCM, iv)
cipher.update(b'header')
ciphertext  = bytearray(cipher.encrypt(data))
print("Ciphertext: {0}".format(ciphertext.hex()))
Ciphertext: ed1759cf244fa97f87de552c1254b1894d1d...

# Get the MAC tag
tag = cipher.digest()
print("Tag: {0}”.format(tag.hex()))
Tag: 701f3c84e2da10aae4b76c89e9ea8427

# ...continued on next slide…
```

The unique part of the GCM code is the **tag generation** and **verification**.

Note the use of **digest()** to get the authentication tag, which is generated from the ciphertext.

NOTE:
In practice, use something like **get_random_bytes(N)** to get N random bytes

Programming using the GCM Mode

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

# Decrypt the ciphertext
cipher = AES.new(key, AES.MODE_GCM, iv)
cipher.update(b'header')
plaintext = cipher.decrypt(ciphertext)
print("Plaintext: {0}".format(plaintext))
Plaintext: b'The quick 7rown fox jumps over the lazy dog'

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

After feeding the ciphertext to the cipher, we invoke **verify()** to verify whether the tag is still valid.

Experiment - GCM Mode

What happens if we modify the ciphertext by changing the 10th byte to (0x00), then decrypt the modified ciphertext and try to verify the tag?

```
$ ./enc_gcm.py
```

Experiment - GCM Mode

What happens if we modify the ciphertext by changing the 10th byte to (0x00), then decrypt the modified ciphertext and try to verify the tag?

```
$ ./enc_gcm.py
Ciphertext: ed1759cf244fa97f87de552c1254b1894d1dad83d8f...a11d
Tag: 701f3c84e2da10aae4b76c89e9ea8427
Plaintext: b'The quick 7rown fox jumps over the lazy dog'
*** Authentication failed ***
```

Check out the docs for tips on more realistic deployment of encryption/decryption and tag verification code:
https://pycryptodome.readthedocs.io/en/latest/src/cipher/modern.html#gcm-mode