Encryption and Decryption and Execution Time measurement script for AES in CBC(128 bit key):

```
start_execution = time.perf_counter()
    key = os.urandom(16)
    iv = os.urandom(16)

    cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
    encryptor = cipher.encryptor()

    start_time = time.perf_counter()

    pad_length = 16 - (len(file_data) % 16)
    padded_data = file_data + bytes([pad_length] * pad_length)
    encrypted_data = encryptor.update(padded_data) + encryptor.finalize()

    encryption_time = time.perf_counter() - start_time

    throughput = (len(file_data) / encryption_time) / (1024 * 1024) if encryption_time > 0 else 0
    execution_time_enc = time.perf_counter() - start_execution


start_execution = time.perf_counter()
    key = base64.b64decode(key_b64)
    iv = base64.b64decode(iv_b64)
    encrypted_data = base64.b64decode(encrypted_data_b64)

    cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
    decryptor = cipher.decryptor()

    start_time = time.perf_counter()
    decrypted_data = decryptor.update(encrypted_data) + decryptor.finalize()
    decryption_time = time.perf_counter() - start_time

    execution_time_dec = time.perf_counter() - start_execution
    execution_time_aes_cbc = execution_time_enc + execution_time_dec
```

Avalanche effect of AES-CBC (128 bit key):

```
def compute_avalanche_effect(original_text, modified_text, key, iv):

  backend = default_backend()
```

```
cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=backend)
encryptor = cipher.encryptor()
original_ciphertext = encryptor.update(original_text) + encryptor.finalize()


cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=backend)
encryptor = cipher.encryptor()
modified_ciphertext = encryptor.update(modified_text) + encryptor.finalize()


bit_diff = xor_bytes(original_ciphertext, modified_ciphertext)
total_bits = len(original_ciphertext) * 8
avalanche_percentage = (bit_diff / total_bits) * 100

modified_plaintext = bytearray(original_plaintext)
modified_plaintext[0] ^= 0b00000001 #flips the least significant bit

return avalanche_percentage, original_ciphertext, modified_ciphertext


Entropy of AES-CBC (128 bit key):
def calculate_entropy(data):

if len(data) == 0:
    print("Warning: Data is empty!")
    return 0.0

byte_frequencies = Counter(data)
total_bytes = len(data)
entropy = 0.0
for freq in byte_frequencies.values():
    probability = freq / total_bytes
    entropy -= probability * math.log2(probability)
return entropy
```