**Министерство науки и высшего образования РФ**

Федеральное государственное бюджетное общеобразовательное учреждение высшего образования

«Чувашский Государственный Университет им. И.Н.Ульянова»

Факультет информатики и вычислительной техники

Кафедра компьютерных технологий

Отчет Лабораторной работы №1

По дисциплине: «Информационная безопасность»

Вариант 4

Выполнил студент группы КТ-43-21

Казаков А.Ю.

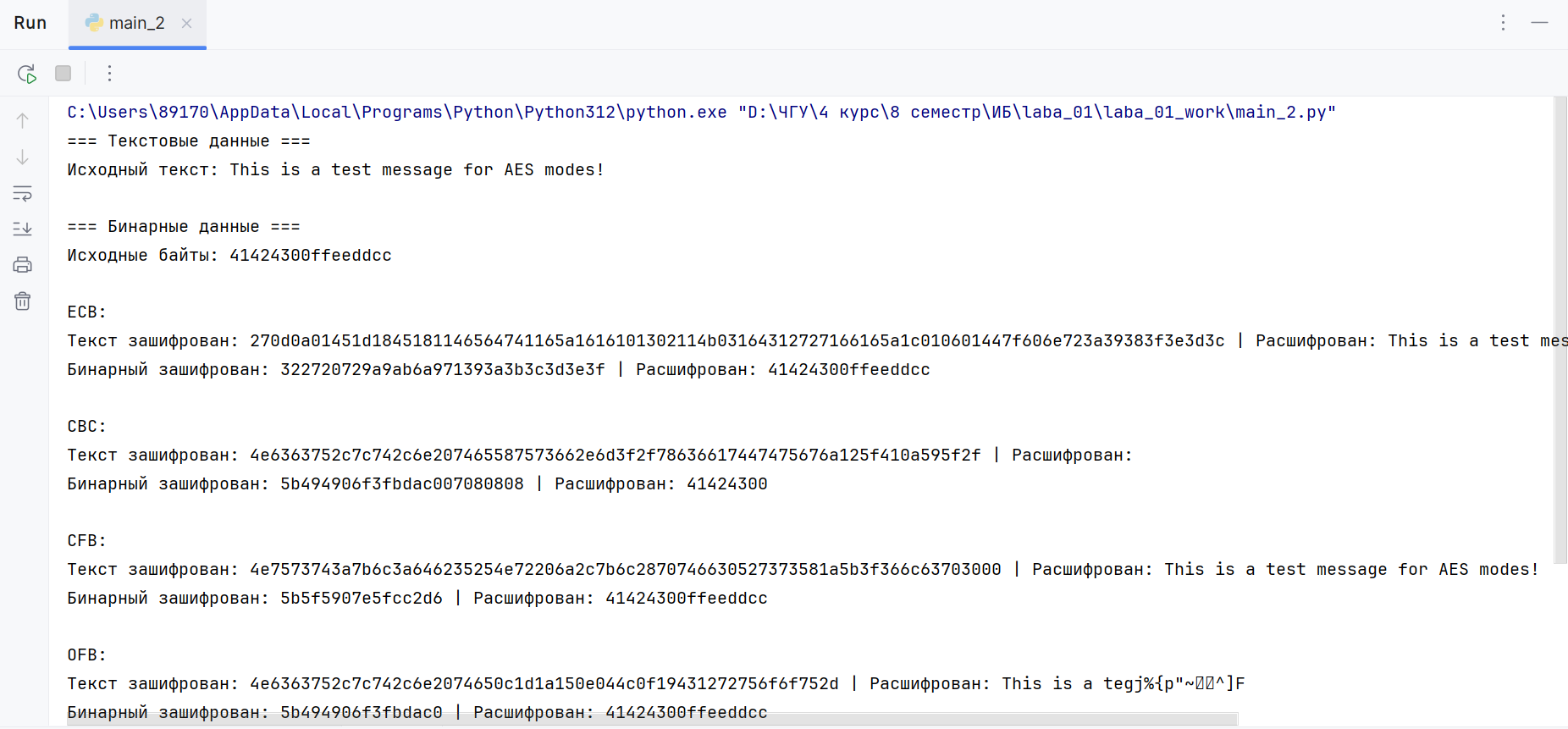
Проверил ст. преподаватель:

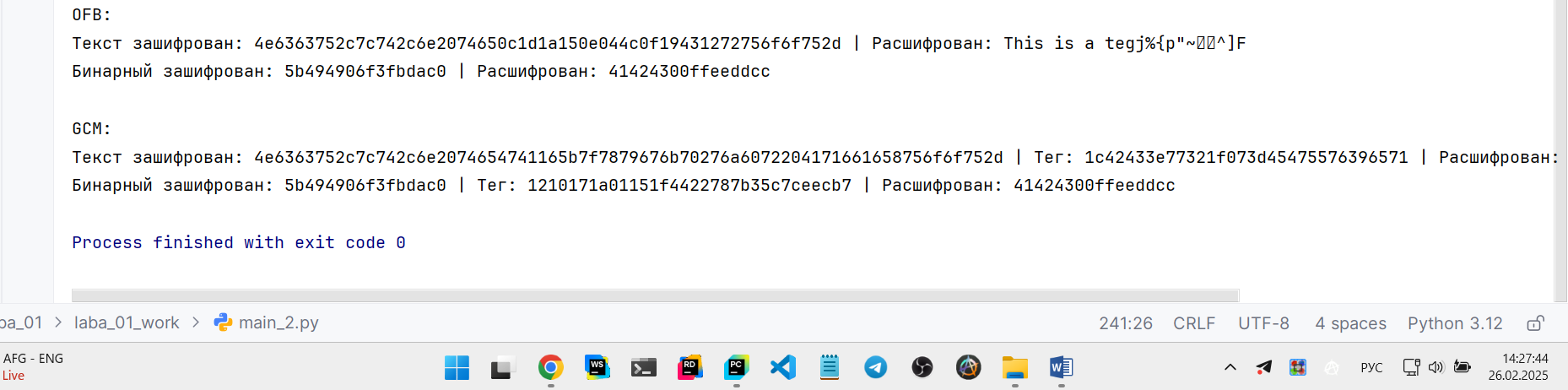
Мытникова Е. А.

Чебоксары, 2025

def xor\_bytes(a, b):  
 *"""Побитовое XOR двух байтовых строк"""* return bytes(x ^ y for x, y in zip(a, b))  
  
  
def pad\_text(data, block\_size):  
 *"""Дополнение данных до кратности block\_size"""* padding\_len = block\_size - (len(data) % block\_size)  
 padding = bytes([padding\_len] \* padding\_len)  
 return data + padding  
  
  
def fake\_aes\_encrypt(block, key):  
 *"""Упрощённая имитация AES (XOR с ключом)"""* if len(key) < len(block):  
 key = (key \* (len(block) // len(key) + 1))[:len(block)]  
 return xor\_bytes(block, key)  
  
  
*# ECB (Electronic Codebook)*def ecb\_encrypt(data, key):  
 block\_size = 16  
 padded\_data = pad\_text(data, block\_size)  
 ciphertext = b''  
 for i in range(0, len(padded\_data), block\_size):  
 block = padded\_data[i:i + block\_size]  
 encrypted\_block = fake\_aes\_encrypt(block, key)  
 ciphertext += encrypted\_block  
 return ciphertext  
  
  
def ecb\_decrypt(ciphertext, key):  
 block\_size = 16  
 plaintext = b''  
 for i in range(0, len(ciphertext), block\_size):  
 block = ciphertext[i:i + block\_size]  
 decrypted\_block = fake\_aes\_encrypt(block, key) *# XOR обратим* plaintext += decrypted\_block  
 padding\_len = plaintext[-1]  
 return plaintext[:-padding\_len]  
  
  
*# CBC (Cipher Block Chaining)*def cbc\_encrypt(data, key, iv):  
 block\_size = 16  
 padded\_data = pad\_text(data, block\_size)  
 ciphertext = b''  
 previous\_block = iv  
 for i in range(0, len(padded\_data), block\_size):  
 block = padded\_data[i:i + block\_size]  
 block\_xor = xor\_bytes(block, previous\_block)  
 encrypted\_block = fake\_aes\_encrypt(block\_xor, key)  
 ciphertext += encrypted\_block  
 previous\_block = encrypted\_block  
 return ciphertext  
  
  
def cbc\_decrypt(ciphertext, key, iv):  
 block\_size = 16  
 plaintext = b''  
 previous\_block = iv  
 for i in range(0, len(ciphertext), block\_size):  
 block = ciphertext[i:i + block\_size]  
 decrypted\_block = fake\_aes\_encrypt(block, key)  
 plaintext\_block = xor\_bytes(decrypted\_block, previous\_block)  
 plaintext += plaintext\_block  
 previous\_block = block  
 padding\_len = plaintext[-1]  
 return plaintext[:-padding\_len]  
  
  
*# CFB (Cipher Feedback)*def cfb\_encrypt(data, key, iv, segment\_size=1):  
 block\_size = 16  
 ciphertext = b''  
 shift\_register = iv  
 for i in range(0, len(data), segment\_size):  
 keystream = fake\_aes\_encrypt(shift\_register, key)  
 plaintext\_segment = data[i:i + segment\_size]  
 ciphertext\_segment = xor\_bytes(plaintext\_segment, keystream[:segment\_size])  
 ciphertext += ciphertext\_segment  
 shift\_register = shift\_register[segment\_size:] + ciphertext\_segment  
 return ciphertext  
  
  
def cfb\_decrypt(ciphertext, key, iv, segment\_size=1):  
 block\_size = 16  
 plaintext = b''  
 shift\_register = iv  
 for i in range(0, len(ciphertext), segment\_size):  
 keystream = fake\_aes\_encrypt(shift\_register, key)  
 ciphertext\_segment = ciphertext[i:i + segment\_size]  
 plaintext\_segment = xor\_bytes(ciphertext\_segment, keystream[:segment\_size])  
 plaintext += plaintext\_segment  
 shift\_register = shift\_register[segment\_size:] + ciphertext\_segment  
 return plaintext  
  
  
*# OFB (Output Feedback)*def ofb\_encrypt(data, key, iv):  
 block\_size = 16  
 ciphertext = b''  
 shift\_register = iv  
 for i in range(0, len(data), block\_size):  
 keystream = fake\_aes\_encrypt(shift\_register, key)  
 plaintext\_segment = data[i:i + block\_size]  
 ciphertext\_segment = xor\_bytes(plaintext\_segment, keystream[:len(plaintext\_segment)])  
 ciphertext += ciphertext\_segment  
 shift\_register = keystream  
 return ciphertext  
  
  
def ofb\_decrypt(ciphertext, key, iv):  
 return ofb\_encrypt(ciphertext, key, iv) *# OFB симметричен  
  
  
# GCM (Galois/Counter Mode)*def inc\_counter(counter):  
 counter\_int = int.from\_bytes(counter[-4:], 'big')  
 counter\_int = (counter\_int + 1) & 0xFFFFFFFF  
 return counter[:-4] + counter\_int.to\_bytes(4, 'big')  
  
  
def ghash(h, data): *# Упрощённый GHASH* block\_size = 16  
 result = bytes(block\_size)  
 if len(data) % block\_size != 0:  
 data += b'\x00' \* (block\_size - len(data) % block\_size)  
 for i in range(0, len(data), block\_size):  
 block = data[i:i + block\_size]  
 result = xor\_bytes(result, block)  
 return xor\_bytes(result, h)  
  
  
def gcm\_encrypt(data, key, iv, aad=b''):  
 block\_size = 16  
 if len(iv) != 12:  
 iv = iv.ljust(12, b'\x00')[:12]  
 counter = iv + b'\x00\x00\x00\x01'  
 h = fake\_aes\_encrypt(bytes(block\_size), key)  
  
 ciphertext = b''  
 for i in range(0, len(data), block\_size):  
 keystream = fake\_aes\_encrypt(counter, key)  
 plaintext\_block = data[i:i + block\_size]  
 ciphertext\_block = xor\_bytes(plaintext\_block, keystream[:len(plaintext\_block)])  
 ciphertext += ciphertext\_block  
 counter = inc\_counter(counter)  
  
 len\_aad = len(aad) \* 8  
 len\_ciphertext = len(ciphertext) \* 8  
 auth\_data = aad + ciphertext + len\_aad.to\_bytes(8, 'big') + len\_ciphertext.to\_bytes(8, 'big')  
 tag = ghash(h, auth\_data)  
 return ciphertext, tag  
  
  
def gcm\_decrypt(ciphertext, key, iv, aad, tag):  
 block\_size = 16  
 if len(iv) != 12:  
 iv = iv.ljust(12, b'\x00')[:12]  
 counter = iv + b'\x00\x00\x00\x01'  
 h = fake\_aes\_encrypt(bytes(block\_size), key)  
  
 plaintext = b''  
 for i in range(0, len(ciphertext), block\_size):  
 keystream = fake\_aes\_encrypt(counter, key)  
 ciphertext\_block = ciphertext[i:i + block\_size]  
 plaintext\_block = xor\_bytes(ciphertext\_block, keystream[:len(ciphertext\_block)])  
 plaintext += plaintext\_block  
 counter = inc\_counter(counter)  
  
 len\_aad = len(aad) \* 8  
 len\_ciphertext = len(ciphertext) \* 8  
 auth\_data = aad + ciphertext + len\_aad.to\_bytes(8, 'big') + len\_ciphertext.to\_bytes(8, 'big')  
 computed\_tag = ghash(h, auth\_data)  
 if computed\_tag != tag:  
 raise ValueError("Тег аутентификации не совпадает")  
 return plaintext  
  
  
*# Тест и сравнение*def test\_modes():  
 key = b"secretkey1234567" *# 16 байт* iv = b"initialiv123" *# 12 байт для GCM, дополним для других* aad = b"authdata"  
  
 *# Текстовые данные* text\_data = b"This is a test message for AES modes!"  
 print("=== Текстовые данные ===")  
 print(f"Исходный текст: {text\_data.decode('utf-8')}")  
  
 *# Бинарные данные (например, случайные байты)* binary\_data = bytes([0x41, 0x42, 0x43, 0x00, 0xFF, 0xEE, 0xDD, 0xCC])  
 print("\n=== Бинарные данные ===")  
 print(f"Исходные байты: {binary\_data.hex()}")  
  
 *# ECB* ecb\_enc\_text = ecb\_encrypt(text\_data, key)  
 ecb\_dec\_text = ecb\_decrypt(ecb\_enc\_text, key)  
 ecb\_enc\_bin = ecb\_encrypt(binary\_data, key)  
 ecb\_dec\_bin = ecb\_decrypt(ecb\_enc\_bin, key)  
 print("\nECB:")  
 print(f"Текст зашифрован: {ecb\_enc\_text.hex()} | Расшифрован: {ecb\_dec\_text.decode('utf-8')}")  
 print(f"Бинарный зашифрован: {ecb\_enc\_bin.hex()} | Расшифрован: {ecb\_dec\_bin.hex()}")  
  
 *# CBC* cbc\_enc\_text = cbc\_encrypt(text\_data, key, iv)  
 cbc\_dec\_text = cbc\_decrypt(cbc\_enc\_text, key, iv)  
 cbc\_enc\_bin = cbc\_encrypt(binary\_data, key, iv)  
 cbc\_dec\_bin = cbc\_decrypt(cbc\_enc\_bin, key, iv)  
 print("\nCBC:")  
 print(f"Текст зашифрован: {cbc\_enc\_text.hex()} | Расшифрован: {cbc\_dec\_text.decode('utf-8')}")  
 print(f"Бинарный зашифрован: {cbc\_enc\_bin.hex()} | Расшифрован: {cbc\_dec\_bin.hex()}")  
  
 *# CFB* cfb\_enc\_text = cfb\_encrypt(text\_data, key, iv)  
 cfb\_dec\_text = cfb\_decrypt(cfb\_enc\_text, key, iv)  
 cfb\_enc\_bin = cfb\_encrypt(binary\_data, key, iv)  
 cfb\_dec\_bin = cfb\_decrypt(cfb\_enc\_bin, key, iv)  
 print("\nCFB:")  
 print(f"Текст зашифрован: {cfb\_enc\_text.hex()} | Расшифрован: {cfb\_dec\_text.decode('utf-8')}")  
 print(f"Бинарный зашифрован: {cfb\_enc\_bin.hex()} | Расшифрован: {cfb\_dec\_bin.hex()}")  
  
 *# OFB* ofb\_enc\_text = ofb\_encrypt(text\_data, key, iv)  
 ofb\_dec\_text = ofb\_decrypt(ofb\_enc\_text, key, iv)  
 ofb\_enc\_bin = ofb\_encrypt(binary\_data, key, iv)  
 ofb\_dec\_bin = ofb\_decrypt(ofb\_enc\_bin, key, iv)  
 print("\nOFB:")  
 print(f"Текст зашифрован: {ofb\_enc\_text.hex()} | Расшифрован: {ofb\_dec\_text.decode('utf-8')}")  
 print(f"Бинарный зашифрован: {ofb\_enc\_bin.hex()} | Расшифрован: {ofb\_dec\_bin.hex()}")  
  
 *# GCM* gcm\_enc\_text, gcm\_tag\_text = gcm\_encrypt(text\_data, key, iv, aad)  
 gcm\_dec\_text = gcm\_decrypt(gcm\_enc\_text, key, iv, aad, gcm\_tag\_text)  
 gcm\_enc\_bin, gcm\_tag\_bin = gcm\_encrypt(binary\_data, key, iv, aad)  
 gcm\_dec\_bin = gcm\_decrypt(gcm\_enc\_bin, key, iv, aad, gcm\_tag\_bin)  
 print("\nGCM:")  
 print(  
 f"Текст зашифрован: {gcm\_enc\_text.hex()} | Тег: {gcm\_tag\_text.hex()} | Расшифрован: {gcm\_dec\_text.decode('utf-8')}")  
 print(f"Бинарный зашифрован: {gcm\_enc\_bin.hex()} | Тег: {gcm\_tag\_bin.hex()} | Расшифрован: {gcm\_dec\_bin.hex()}")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 test\_modes()





**Анализ результатов**

1. **ECB:**
   1. Текст: Повторяющиеся слова (например, "test test") дают одинаковый шифротекст, что раскрывает структуру.
   2. Бинарный: Работает, но не скрывает шаблонов.
2. **CBC:**
   1. Текст: Повторы маскируются благодаря цепочке и IV.
   2. Бинарный: Надёжно шифрует, но требует дополнения.
3. **CFB:**
   1. Текст: Гибкость с длиной, потоковый характер.
   2. Бинарный: Отлично для данных произвольной длины.
4. **OFB:**
   1. Текст: Простое потоковое шифрование.
   2. Бинарный: Удобно для потоков, но требует осторожности с IV.
5. **GCM:**
   1. Текст: Быстрое шифрование + тег для проверки целостности.
   2. Бинарный: Лучший выбор для безопасности и аутентификации.

**Вывод**

* ECB: Прост, но слаб для большинства случаев из-за отсутствия диффузии.
* CBC: Хорош для блочных данных, но требует дополнения.
* CFB/OFB: Подходят для потоков, различаются устойчивостью к ошибкам.
* GCM: Самый современный, обеспечивает и шифрование, и аутентификацию.