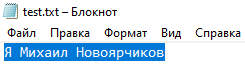
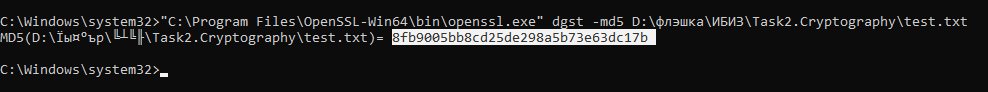
# Криптография

## Хэш функции

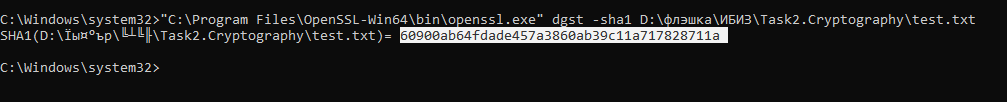
Исходный файл:



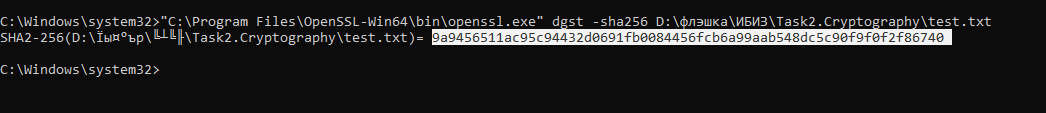
Подсчет хеша с помощью –md5:

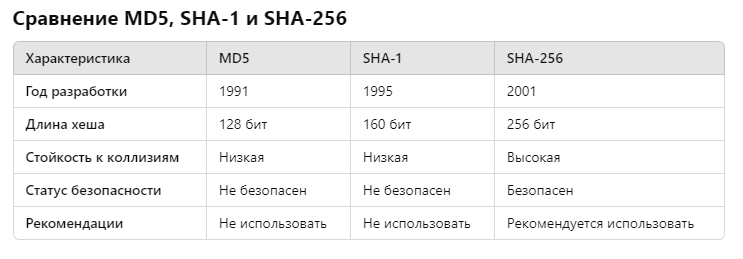


Подсчет хеша с помощью –sha1:

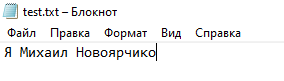


Подсчет хеша с помощью –sha256:

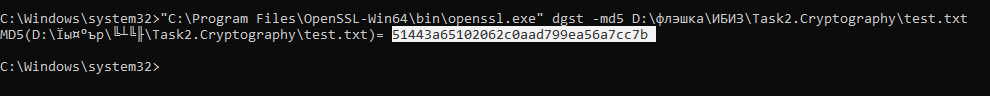




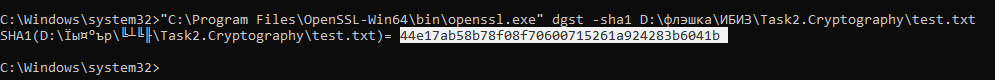
Изменил 1 бит в исходном сообщении:



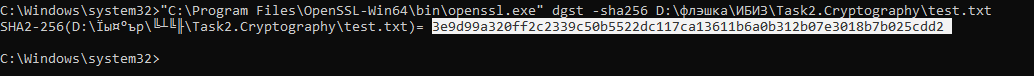
Подсчет хеша с помощью –md5:



Подсчет хеша с помощью –sha1:



Подсчет хеша с помощью –sha256:

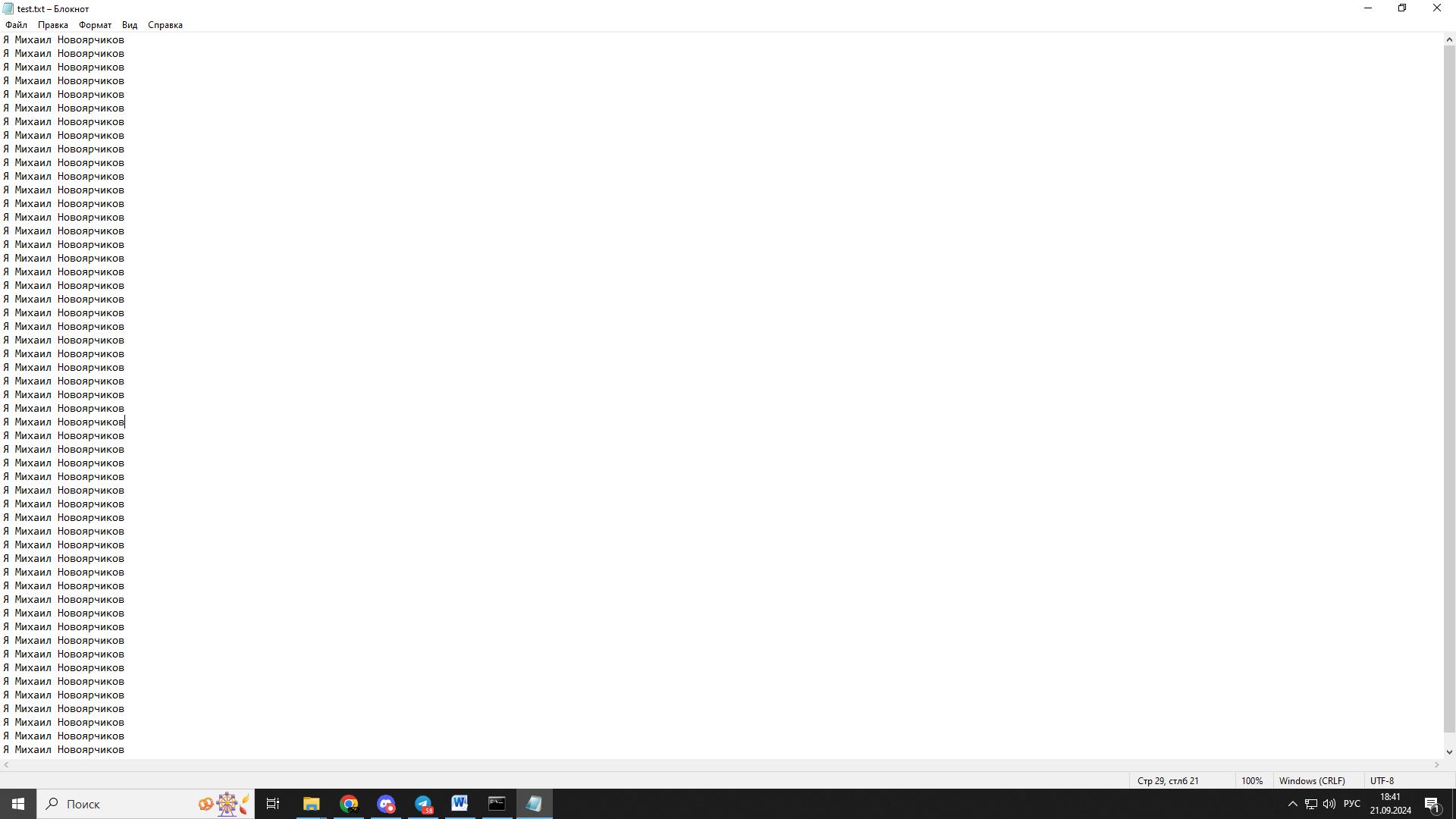


**Эффект лавины** - хорошая хеш-функция характеризуется эффектом лавины: любое небольшое изменение входных данных (например, изменение одного бита) приводит к радикальному изменению выходного хеша. Это значит, что нет прямой зависимости между входными и выходными данными, и даже знание конечного хеша не даёт возможности проследить, как небольшие изменения во входных данных повлияли на выходной результат. (Происходит из-за конкатенации переменных H0-H7, так как каждый блок хэшируется отдельно).

## Симметричное шифрование

Файл размером больше 1кб:

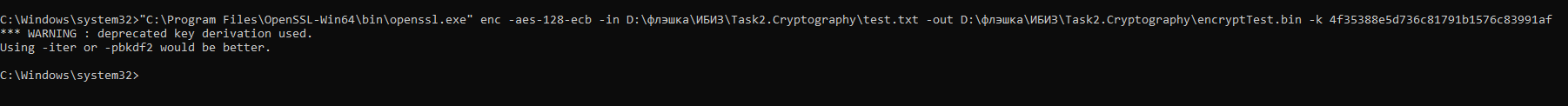




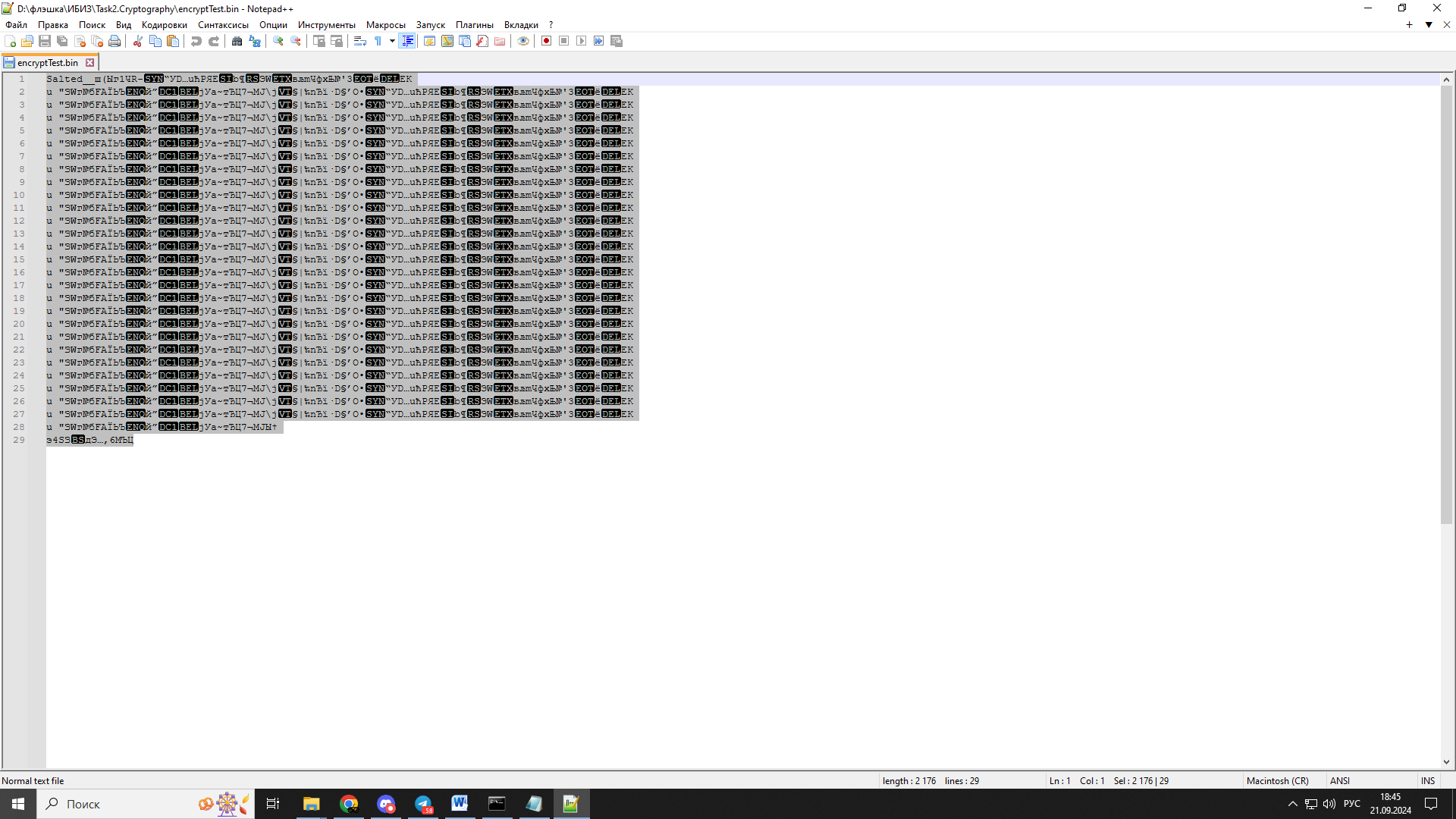
**ECB (Electronic Codebook)**: Самый простой режим, где каждый блок данных шифруется отдельно. Однако он считается небезопасным, так как одинаковые блоки открытого текста будут зашифрованы одинаково.

Команда для зашифровки:

openssl enc -aes-128-ecb -in yourfile.txt -out encrypted\_ecb.bin -k yourpassword

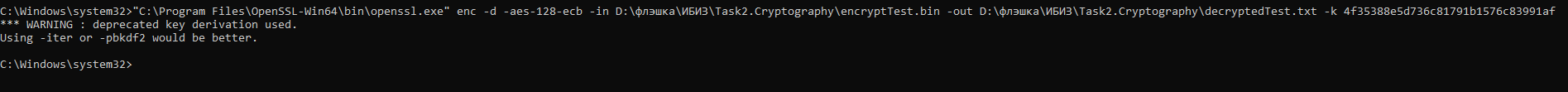


Зашифрованный файл:

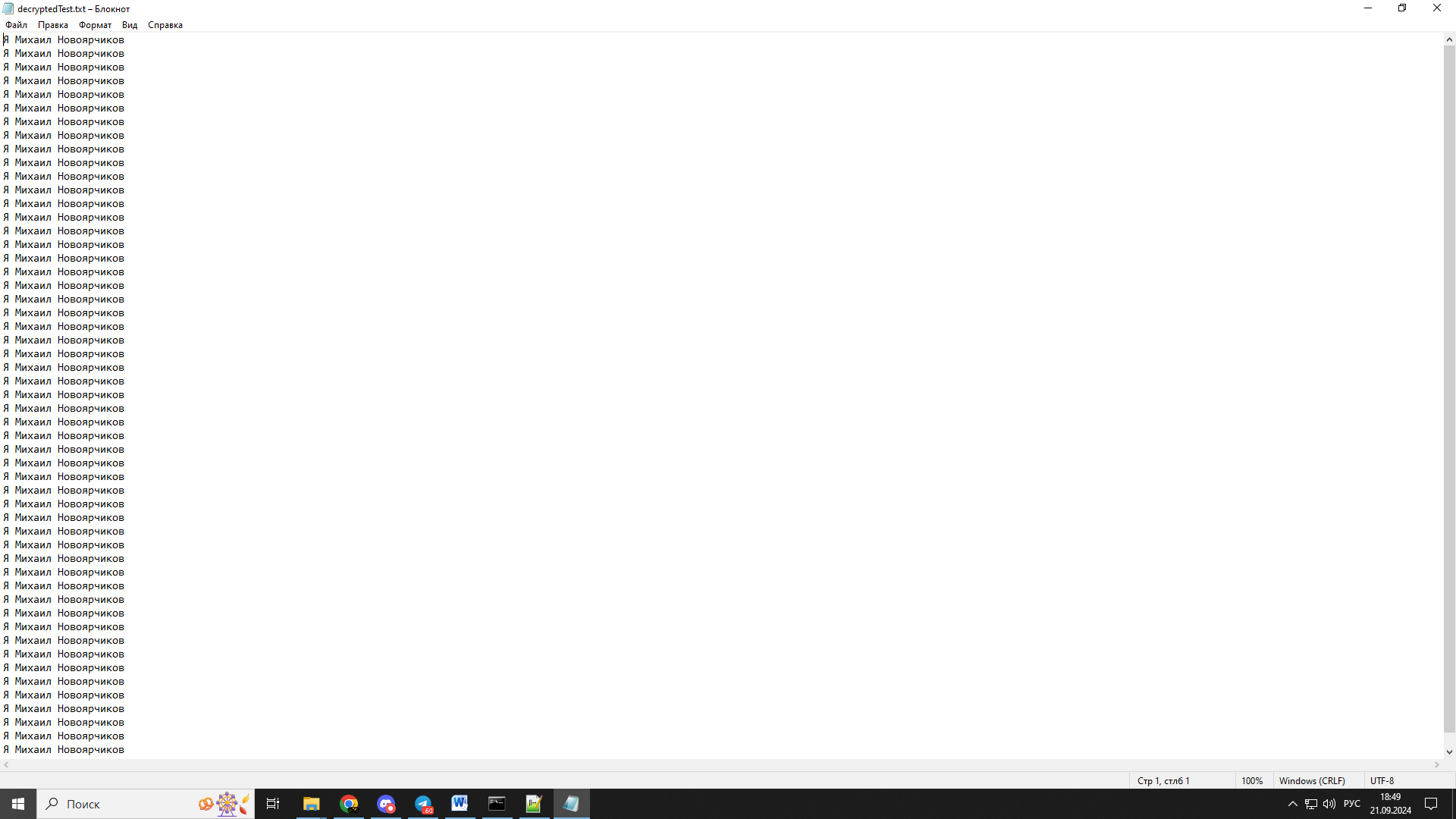


Команда для расшифрования:

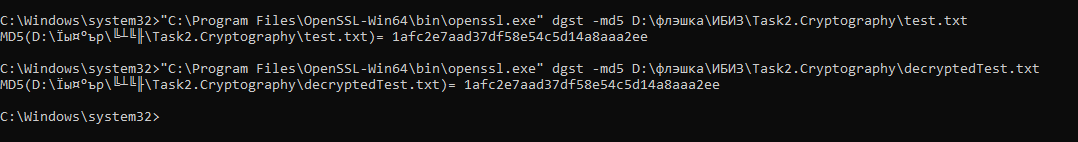
openssl enc -d -aes-128-ecb -in encrypted\_ecb.bin -out decrypted\_ecb.txt -k yourpassword



Расшифрованный файл:



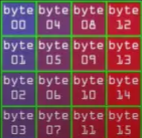
Сравнение хешей – идентичны:



-k – ключ для шифровки.

**Ключ** — это набор битов (чисел), используемый для управления процессом шифрования и расшифровки данных. В симметричных алгоритмах шифрования, таких как AES, один и тот же ключ используется как для шифрования, так и для расшифровки.

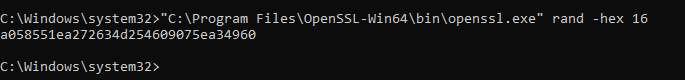
Алгоритм AES работает с **16 байтами данных** за раз. Эти 16 байт образуют **матрицу 4x4** (называемую **State**), в которой каждый элемент — это один байт (8 бит).



В случае с AES у нас в каждом раунде шифрования добавляется RoundKey, который комбинируется с матрицей с помощью операции XOR. А самый первый раундовый ключ создается на основе нашего ключа.

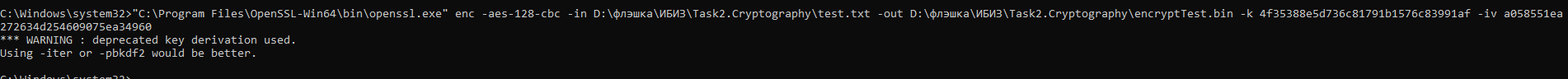
**CBC (Cipher Block Chaining)**: Более безопасный режим, в котором каждый блок данных шифруется с использованием предыдущего зашифрованного блока. Для первого блока используется вектор инициализации (IV).

Генерирую вектор инициализаци IV:

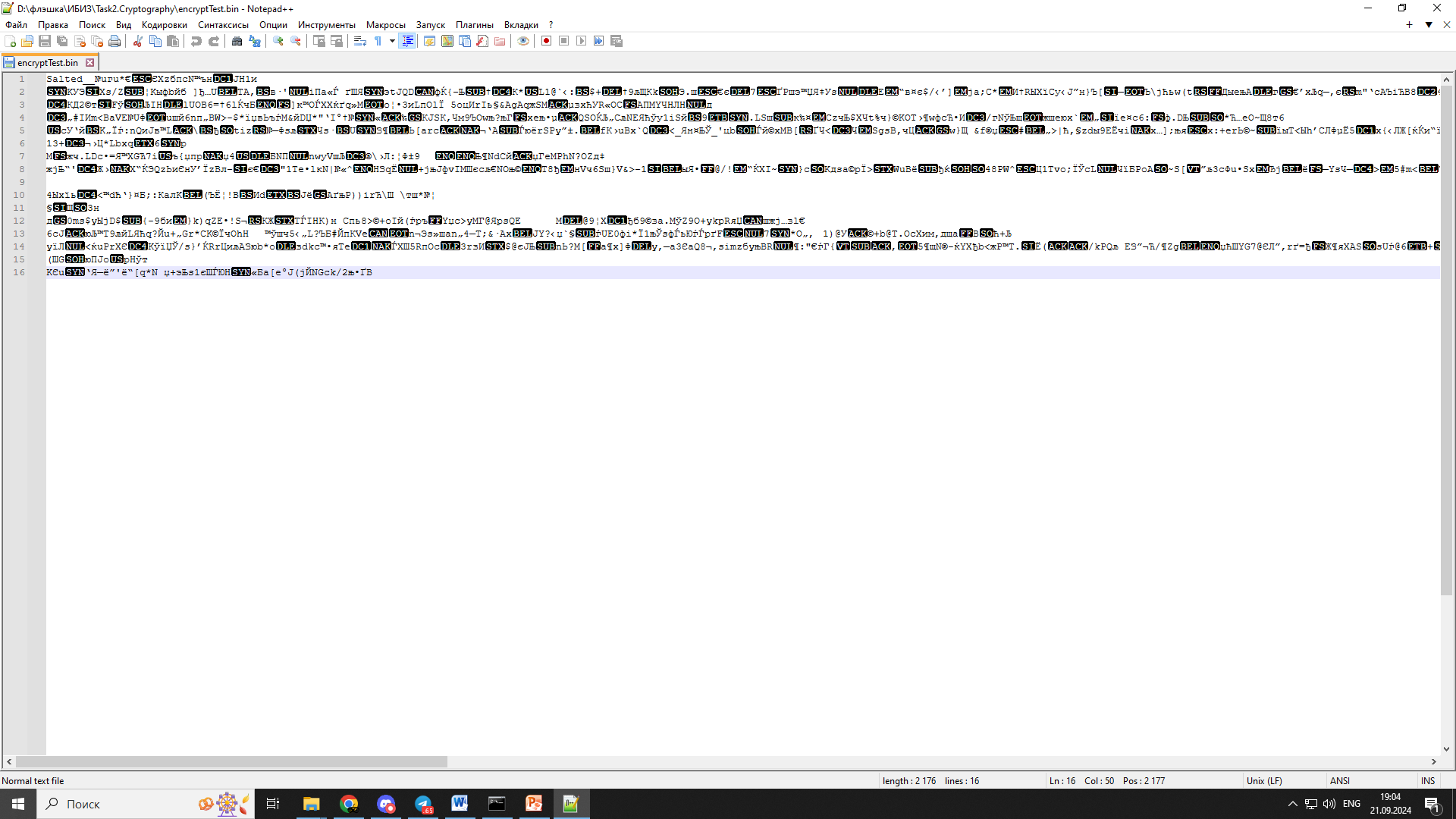


Команда для шифрования:

openssl enc -aes-128-cbc -in yourfile.txt -out encrypted\_cbc.bin -k yourpassword -iv youriv

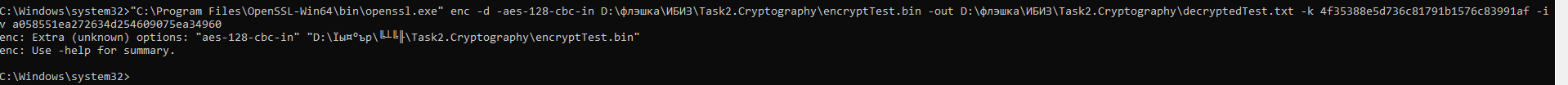


Зашифрованный файл:

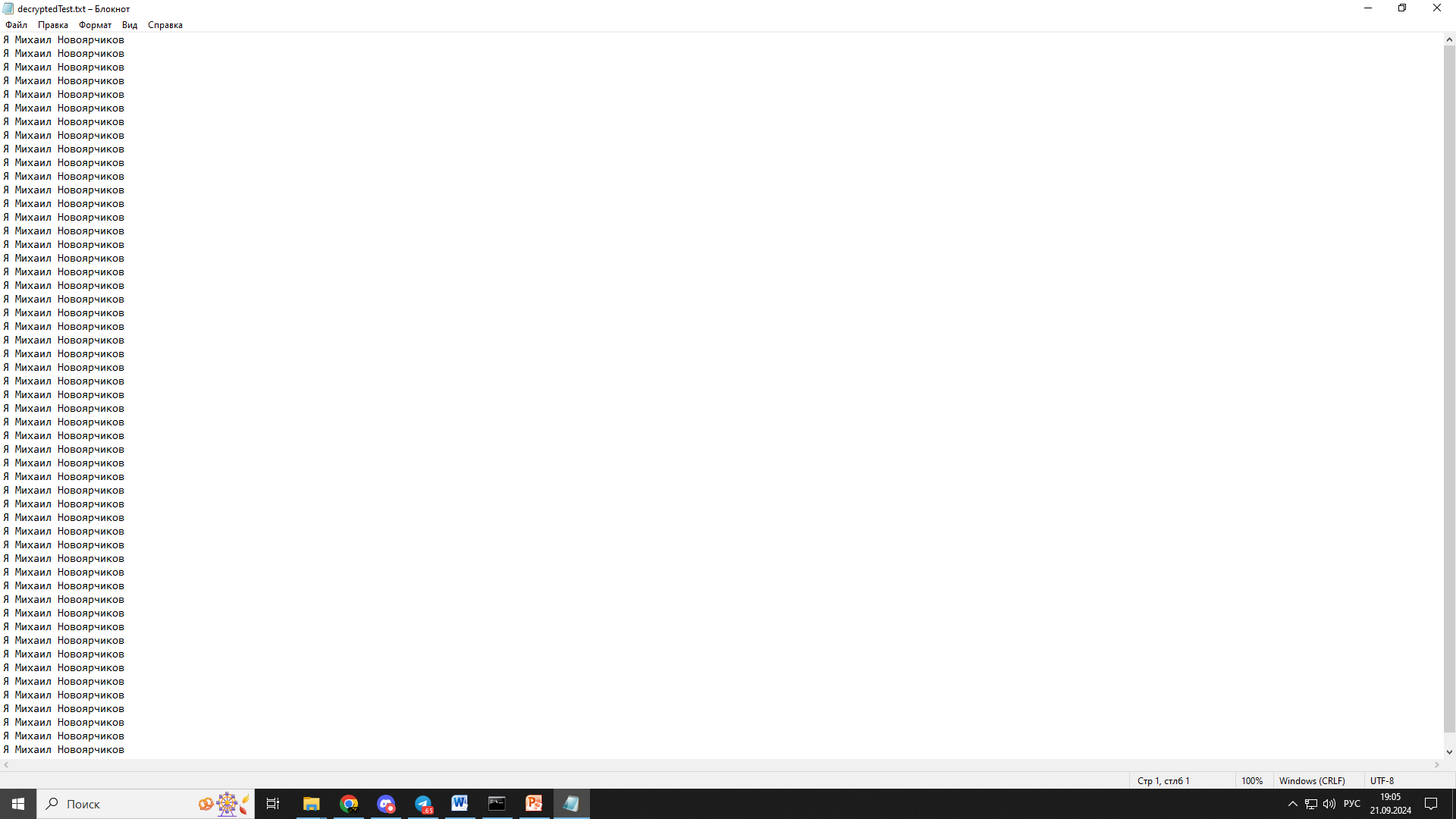


Команда для расшифрования:

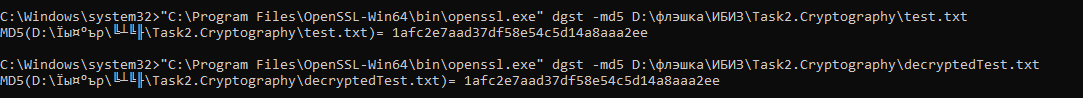
openssl enc -d -aes-128-cbc -in encrypted\_cbc.bin -out decrypted\_cbc.txt -k yourpassword -iv youriv



Расшифрованный файл:



Сравнение хешей – идентичны:



В режиме **CBC** каждый блок данных перед шифрованием комбинируется с предыдущим зашифрованным блоком, чтобы добавить зависимость между блоками и повысить безопасность.

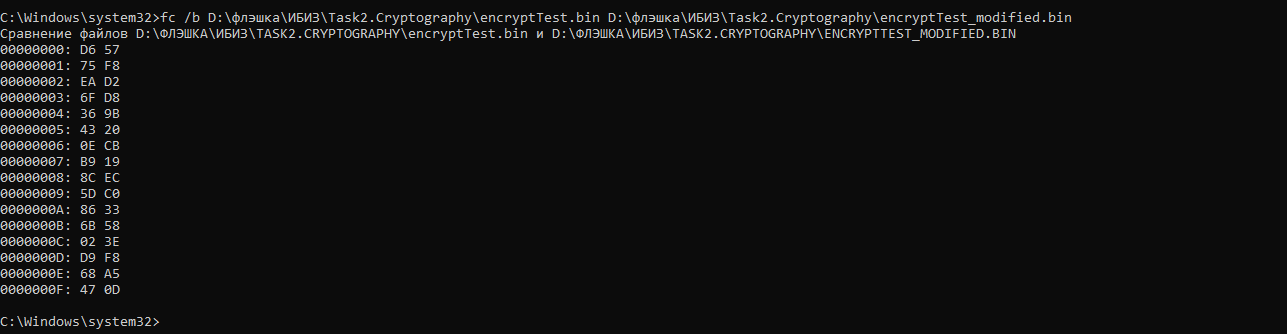
**Шаги шифрования:**

* **Первый блок (B1)**:
  + Перед шифрованием первый блок данных **B1** комбинируется (побитово через операцию XOR) с **вектором инициализации (IV)**. IV — это случайные данные, которые необходимы для шифрования первого блока, поскольку у него нет предшествующего блока.
  + Полученный результат шифруется с использованием ключа AES, и это становится зашифрованным первым блоком **C1** (Ciphertext 1).
* **Второй блок (B2)**:
  + Теперь второй блок данных **B2** комбинируется (побитово через XOR) с **зашифрованным первым блоком C1**.
  + Затем этот результат шифруется с использованием того же ключа AES, и это становится зашифрованным вторым блоком **C2** (Ciphertext 2).

Такой процесс продолжается для каждого следующего блока. Каждый новый блок данных перед шифрованием зависит от предыдущего зашифрованного блока.

Эта цепочка шифрования сохраняется, что делает каждый блок зашифрованных данных зависимым от предыдущих. Если изменить хотя бы один байт в первом блоке, это изменит все последующие зашифрованные блоки.

**Изменение первого бита и сравнение зашифровки ECB:**

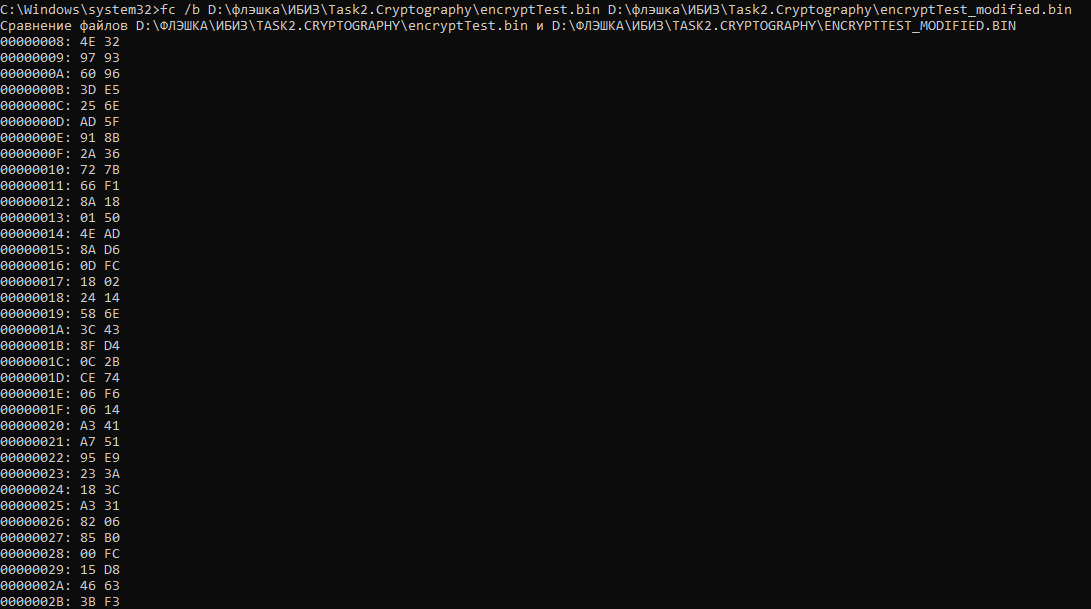


Разница только в первых 16 байтах (первый блок).

Важно:

1. Удаление приводит к изменению размера файла, соответственно вся шифровка изменяется и эксперимент не провести.
2. Нужно использовать –K, тк –k использует соль, из-за которой шифрование каждый раз может быть уникальным

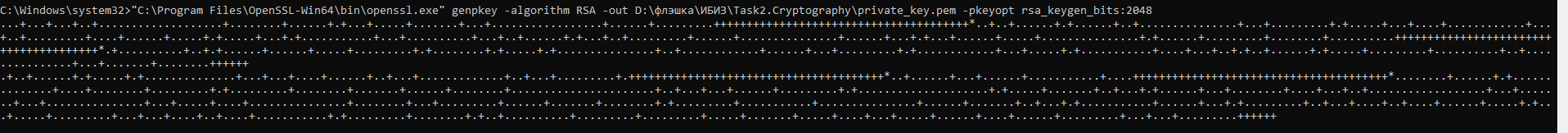
**Изменение первого бита и сравнение зашифровки CBC:**

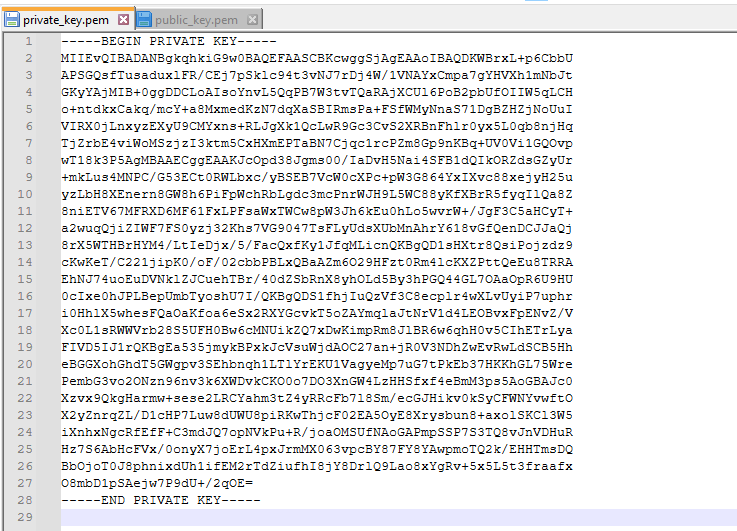


Все блоки различаются за счет вектора инициализации и комбинации шифрования блоков.

## Ассиметричное шифрование

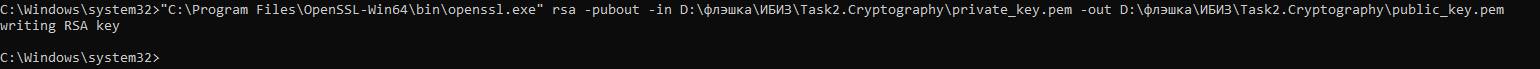
Генерация приватного ключа:

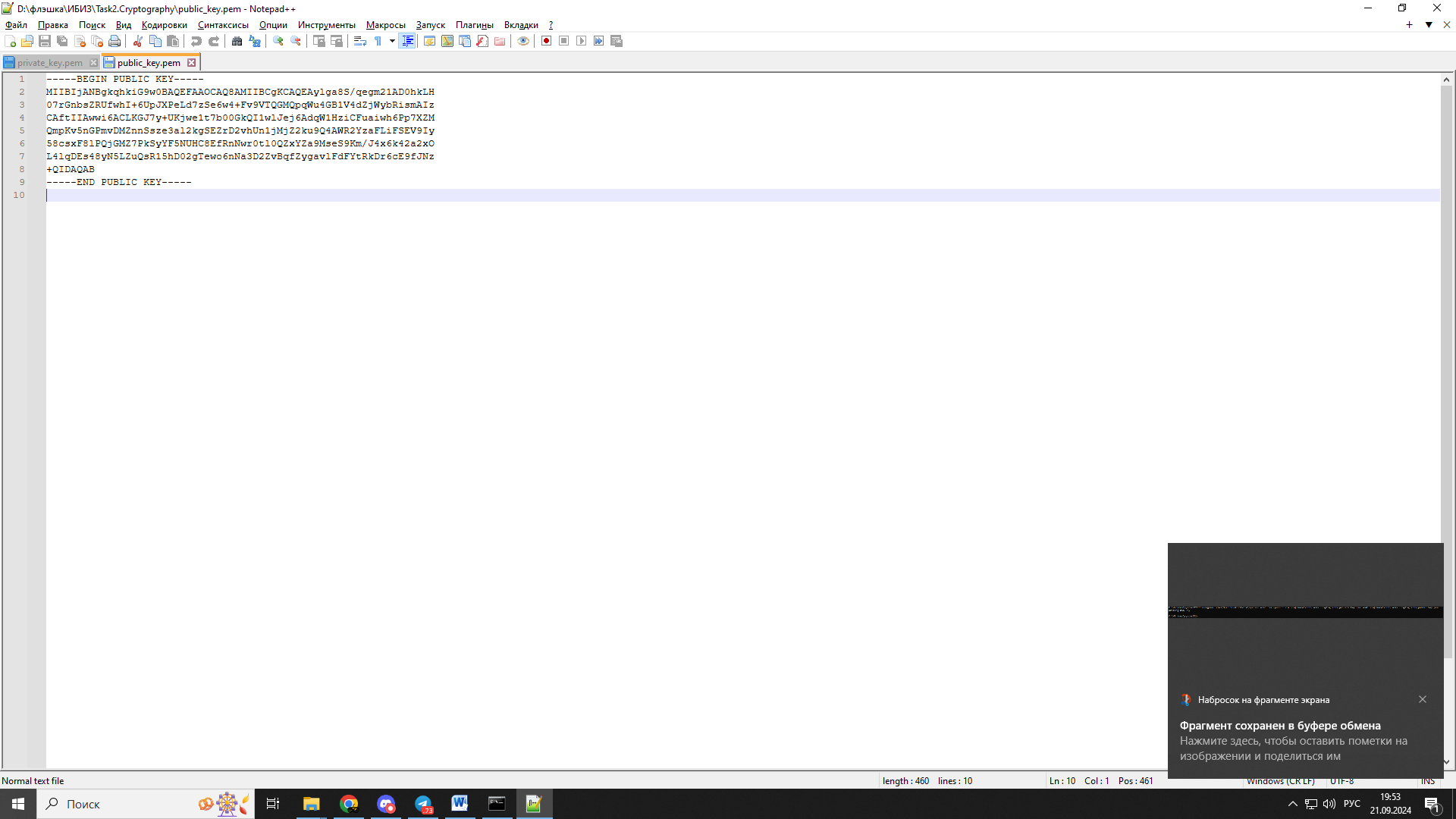
openssl genpkey -algorithm RSA -out private\_key.pem -pkeyopt rsa\_keygen\_bits:2048 



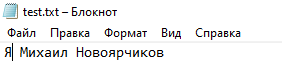
Генерация публичного ключа:

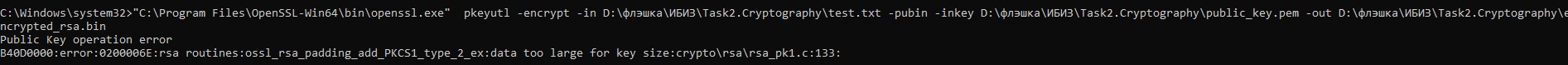
openssl rsa -pubout -in private\_key.pem -out public\_key.pem





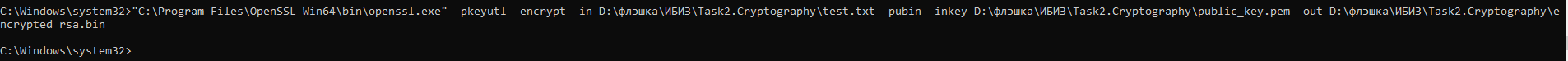
Для шифрования исходный файл был уменьшен в размерах, так как RSA не поддерживал шифрование больших файлов:





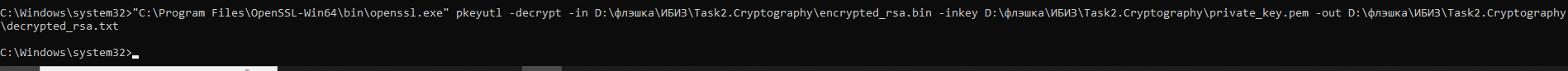
Шифрование публичным ключом:

openssl pkeyutl -encrypt -in yourfile.txt -pubin -inkey public\_key.pem -out encrypted\_rsa.bin

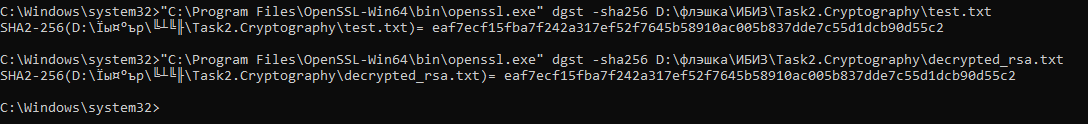


Расшифровка приватным ключом:

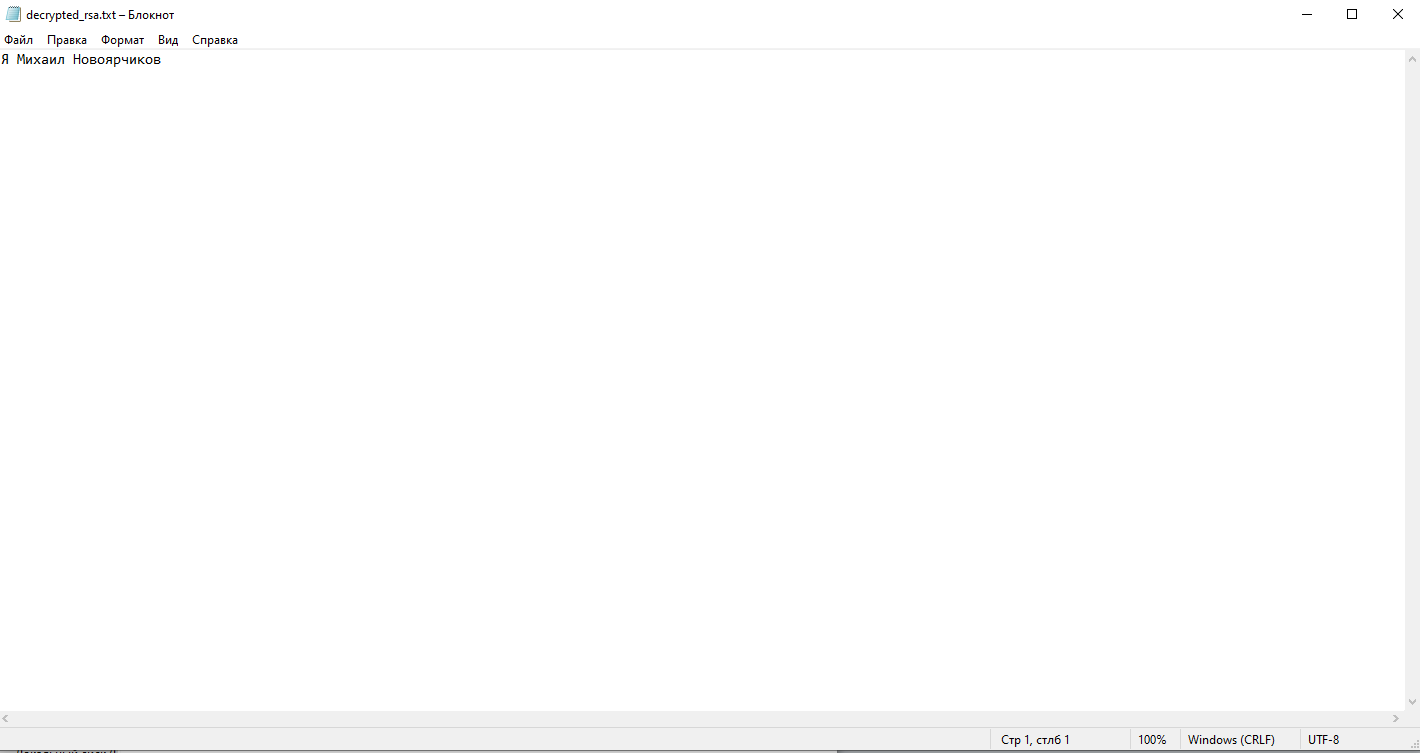
openssl pkeyutl -decrypt -in encrypted\_rsa.bin -inkey private\_key.pem -out decrypted\_rsa.txt



Сравнение хэшей – идентичны:



Расшифрованный файл:

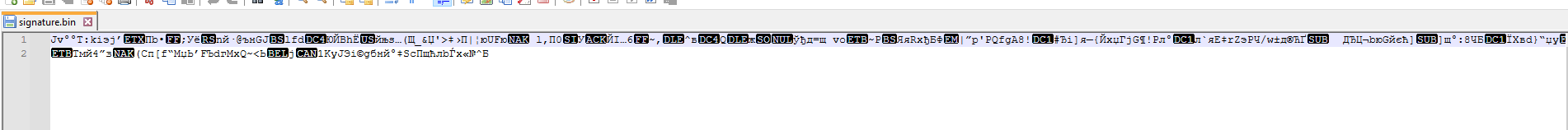


1. **Открытый ключ**: Этот ключ используется для шифрования данных и может быть свободно передан другим пользователям или опубликован. Он не раскрывает информацию, которая может быть использована для расшифровки данных.
2. **Закрытый ключ**: Этот ключ используется для расшифровки данных и должен храниться в тайне. Только владелец закрытого ключа может расшифровать данные, зашифрованные с использованием соответствующего открытого ключа.

## 4. Электронная цифровая подпись (ЭЦП).

Подпись файла с помощью закрытого ключа RSA:

openssl dgst -sha256 -sign private\_key.pem -out signature.bin yourfile.txt 



Проверка подписи файла с помощью публичного ключа RSA:

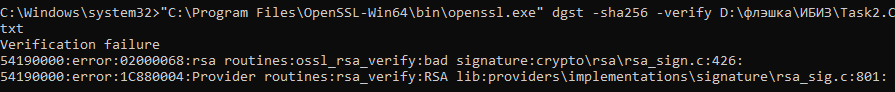
openssl dgst -sha256 -verify public\_key.pem -signature signature.bin yourfile.txt



Проверка прошла успешно.

Изменение 1 байта в исходном файле:



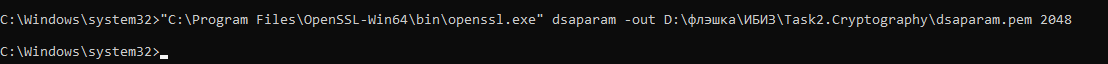


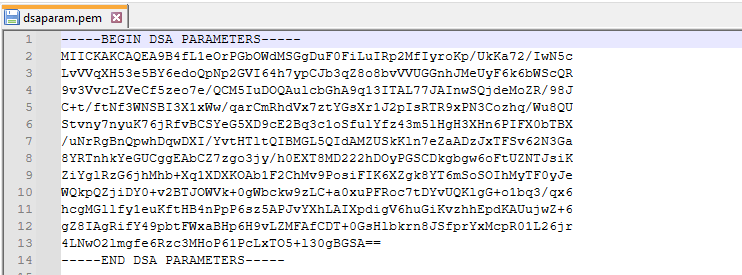
Проверка подписи не прошла, так как хэш исходного сообщения получился другой (пришел битый или модифицированный файл)

**DSA:**

Генерация параметров:

openssl dsaparam -out dsaparam.pem 2048

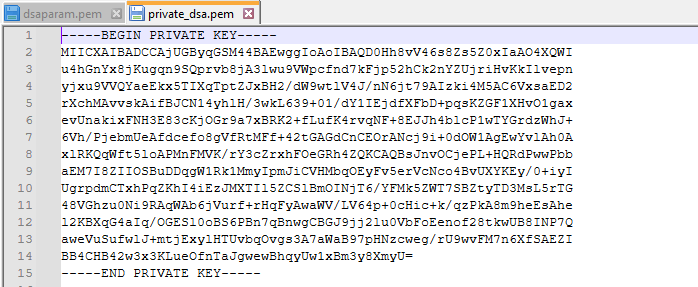




Генерация приватного ключа:

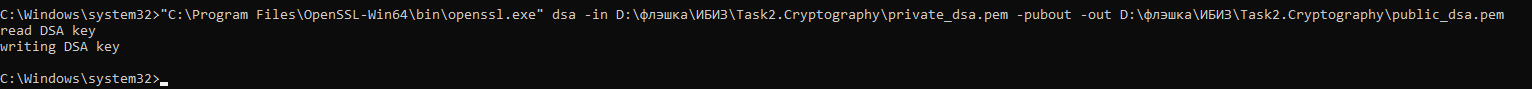
openssl gendsa -out private\_dsa.pem dsaparam.pem





Генерация публичного ключа:

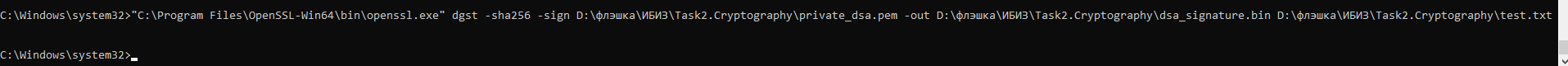
openssl dsa -in private\_dsa.pem -pubout -out public\_dsa.pem





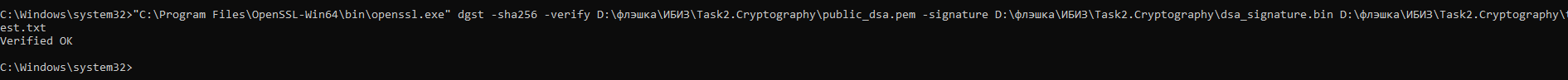
Подпись файла закрытым ключом DSA:

openssl dgst -sha256 -sign private\_dsa.pem -out dsa\_signature.bin yourfile.txt



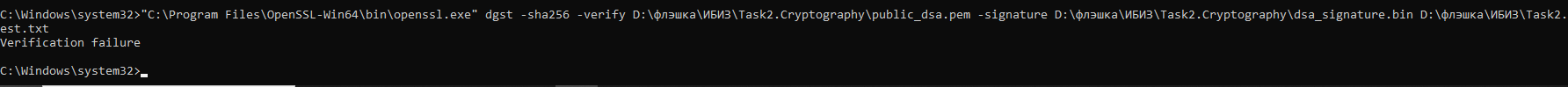
Проверка подписи публичным ключом DSA:

openssl dgst -sha256 -verify public\_dsa.pem -signature dsa\_signature.bin yourfile.txt



Проверка прошла успешно.

После изменения 1 байта проверка не проходит:



Причина все та же, что и в прошлом случае.

**Отличия DSA от RSA**

**1. Назначение и функциональность:**

DSA:

* Предназначен только для создания и проверки цифровых подписей.
* Не поддерживает шифрование или обмен ключами.

RSA:

* Универсальный алгоритм, поддерживающий как шифрование, так и цифровую подпись.
* Используется для шифрования данных, обмена ключами и цифровых подписей.

**2. Алгоритмическая основа:**

DSA:

* Основан на сложности вычисления дискретного логарифма в конечных полях.

RSA:

* Основан на сложности факторизации больших целых чисел.

**3. Производительность:**

Подписание:

* DSA обычно быстрее генерирует подпись по сравнению с RSA при одинаковой длине ключа.

Проверка подписи:

* RSA быстрее проверяет подпись по сравнению с DSA.

4. Размер подписи:

DSA:

* Размер подписи фиксирован и зависит от размера q (обычно меньше, чем у RSA).

RSA:

* Размер подписи равен размеру модуля n (длина ключа), поэтому подпись RSA обычно длиннее.

5. Безопасность и стандарты:

DSA:

* Требует надежного генератора случайных чисел для k.
* Уязвим, если k скомпрометирован или повторно использован.

RSA:

* Безопасность зависит от длины ключа и правильной реализации алгоритма.