Быстрый анализ игрового датафайла

сентября 21, 2016

Хуй знает, что меня побудило на написание этой статьи.

Ни для кого не секрет, что во многих играх используются кастомные бинарные форматы для хранения множества файлов в первичном или сжатом виде.Естественно, что во время загрузки любой игры из этих бинарных файлов происходит считывание и выгрузка в оперативную память всего необходимого.

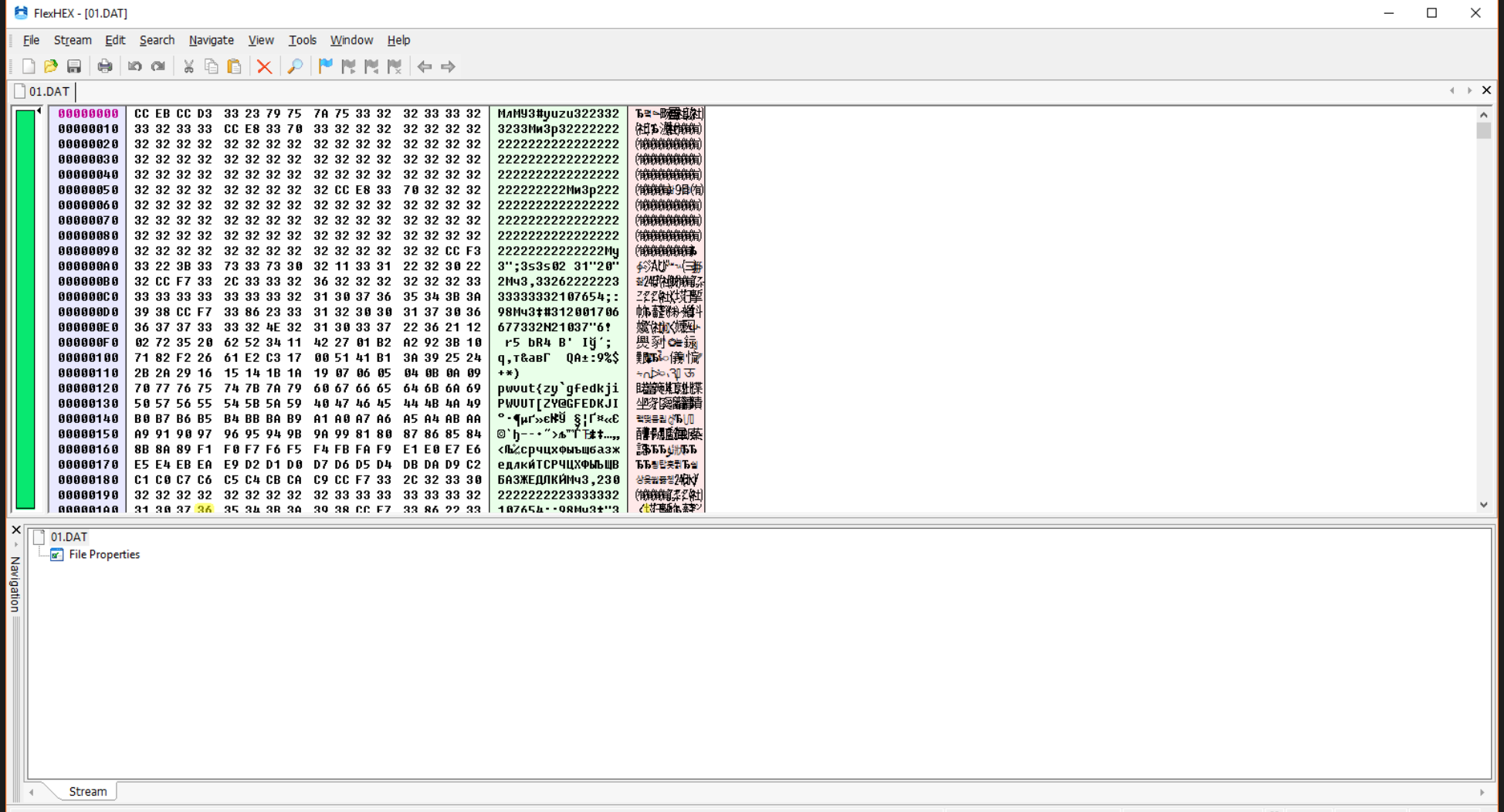
Но как быть, если файл неизвестного формата настолько, что в нем нельзя найти ни TOC, ни форматные сигнатуры ни в начале, ни в конце ?

А очень просто.

Любой зашифрованный или сжатый файл проходит предварительную расшифровку/разжатие в оперативной памяти.Это происходит благодаря вызовам определенного числа функций в самом исполняемом файле или игровом скрипте, но мы не будет реверсить исполняемый файл.Достаточно будет анализа бинарного формата в конкретном случае.

Возьмем демоверсию игры "X - Beyond The Frontier", вышедшей в свет в далеком 1999 году.

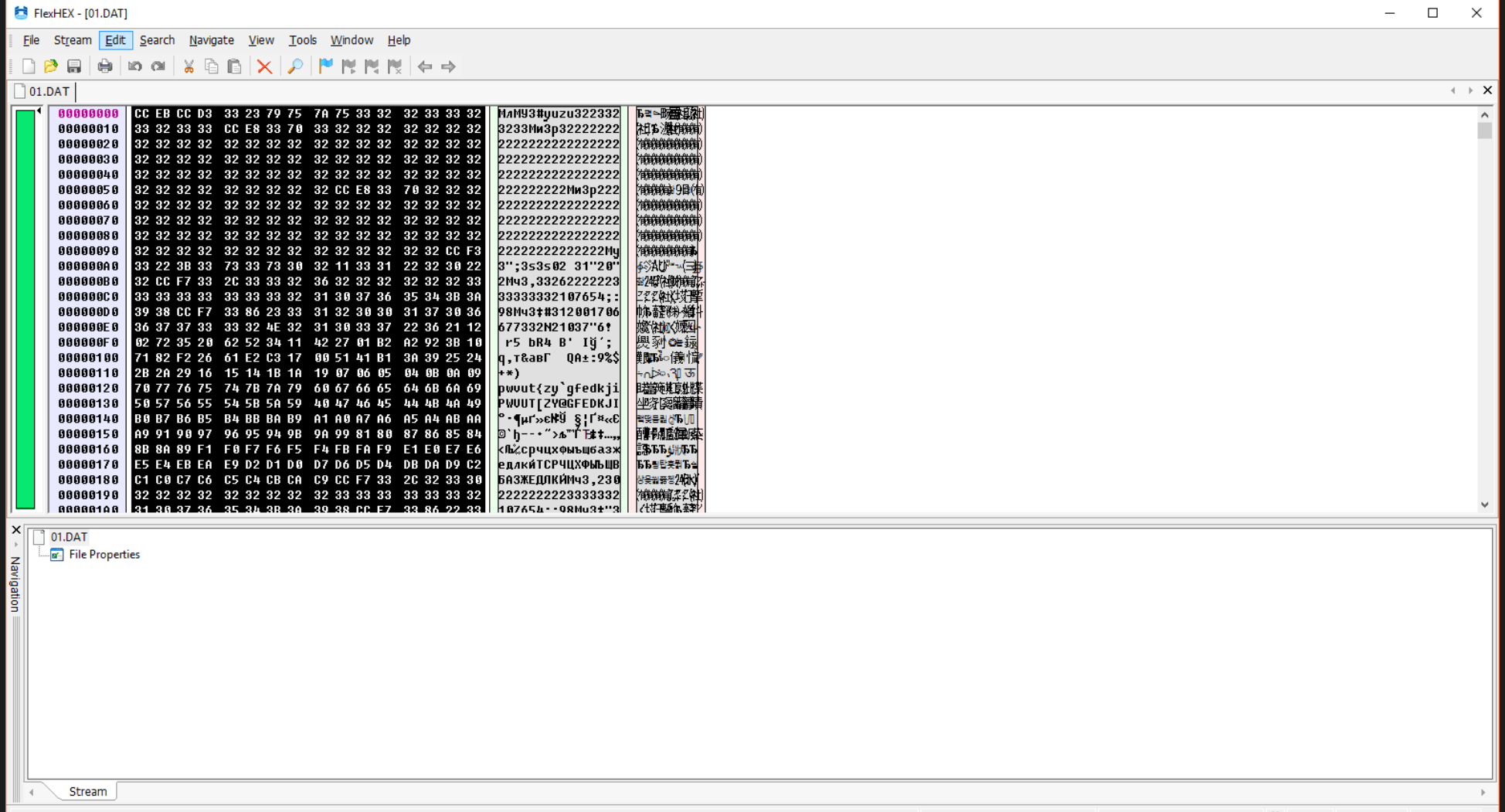
Дропнем в мой любимый hex-редактор файл **01.DAT**



Сразу же можно заметить, что в файле есть повторяющиеся байты, такие как **32h** и менее встречающиеся **33h**.

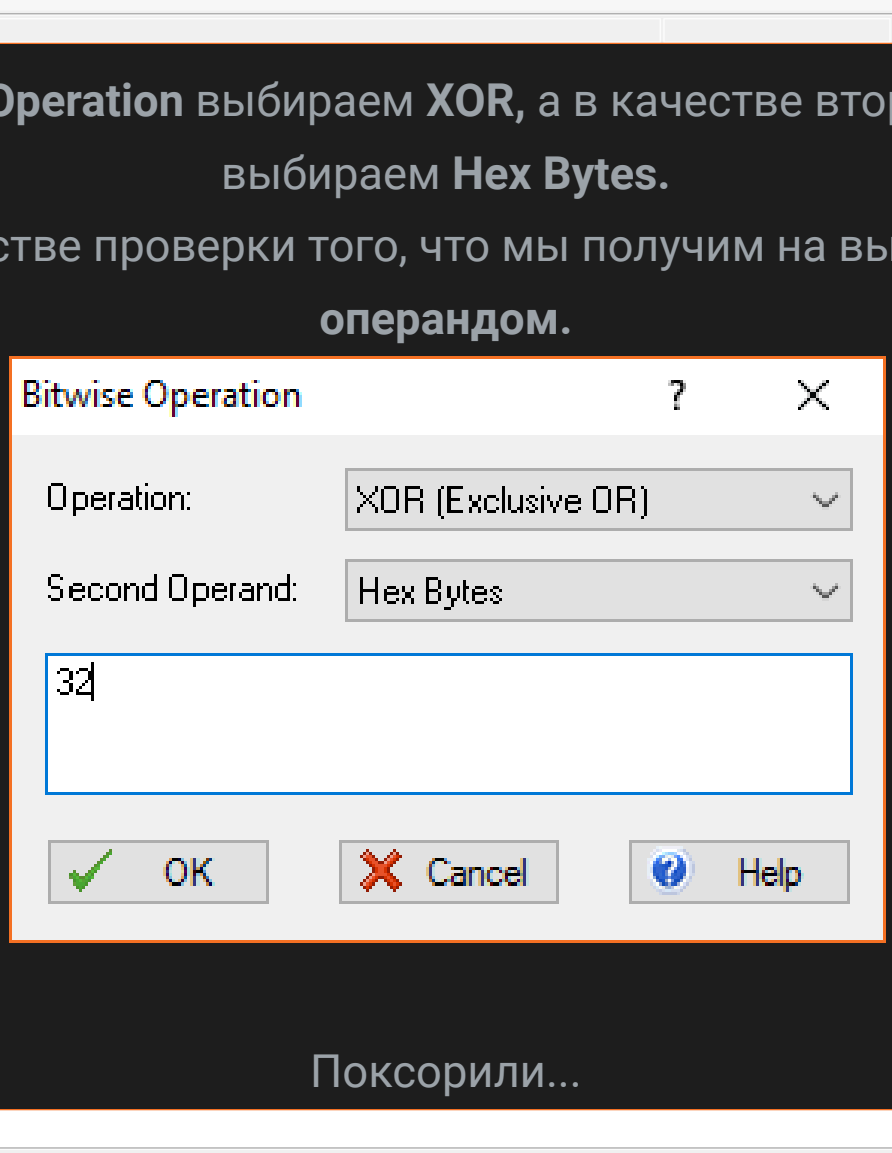
> *Практика показывает, что игровые разработчики не особо запариваются с шифрованием, поэтому перед нами с вероятностью 80% битовая операция XOR, которая встречается довольно часто в различных игровых форматах и не только.*

Проверим.Выделим все байты с помощью **Ctrl+A** и обратимся к меню **Edit**, а затем выберем **Operation -> Bitwise...**

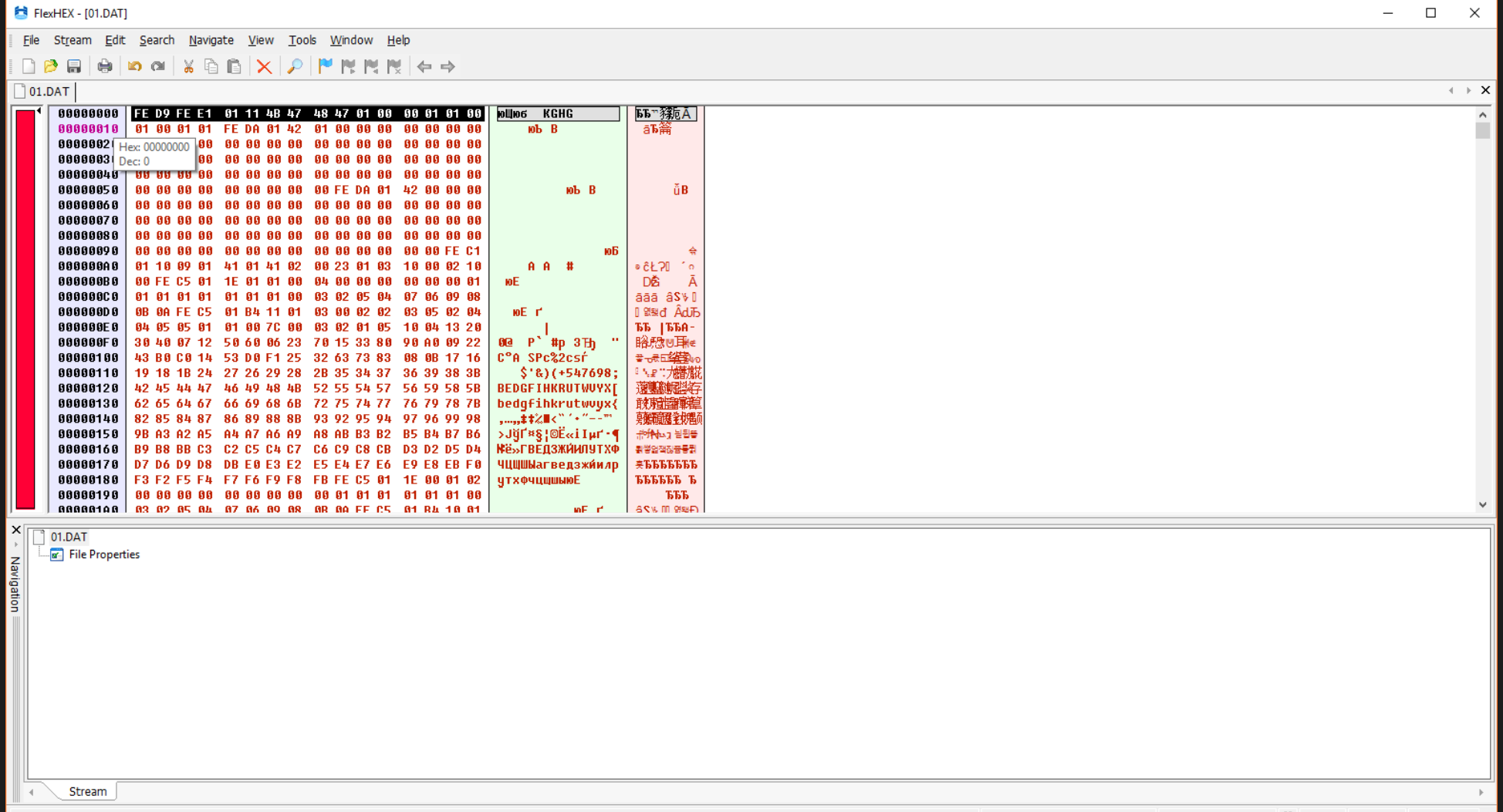


В появившемся окне из списка **Operation** выбираем **XOR,** а в качестве второго операнда во втором списке выбираем **Hex Bytes.**

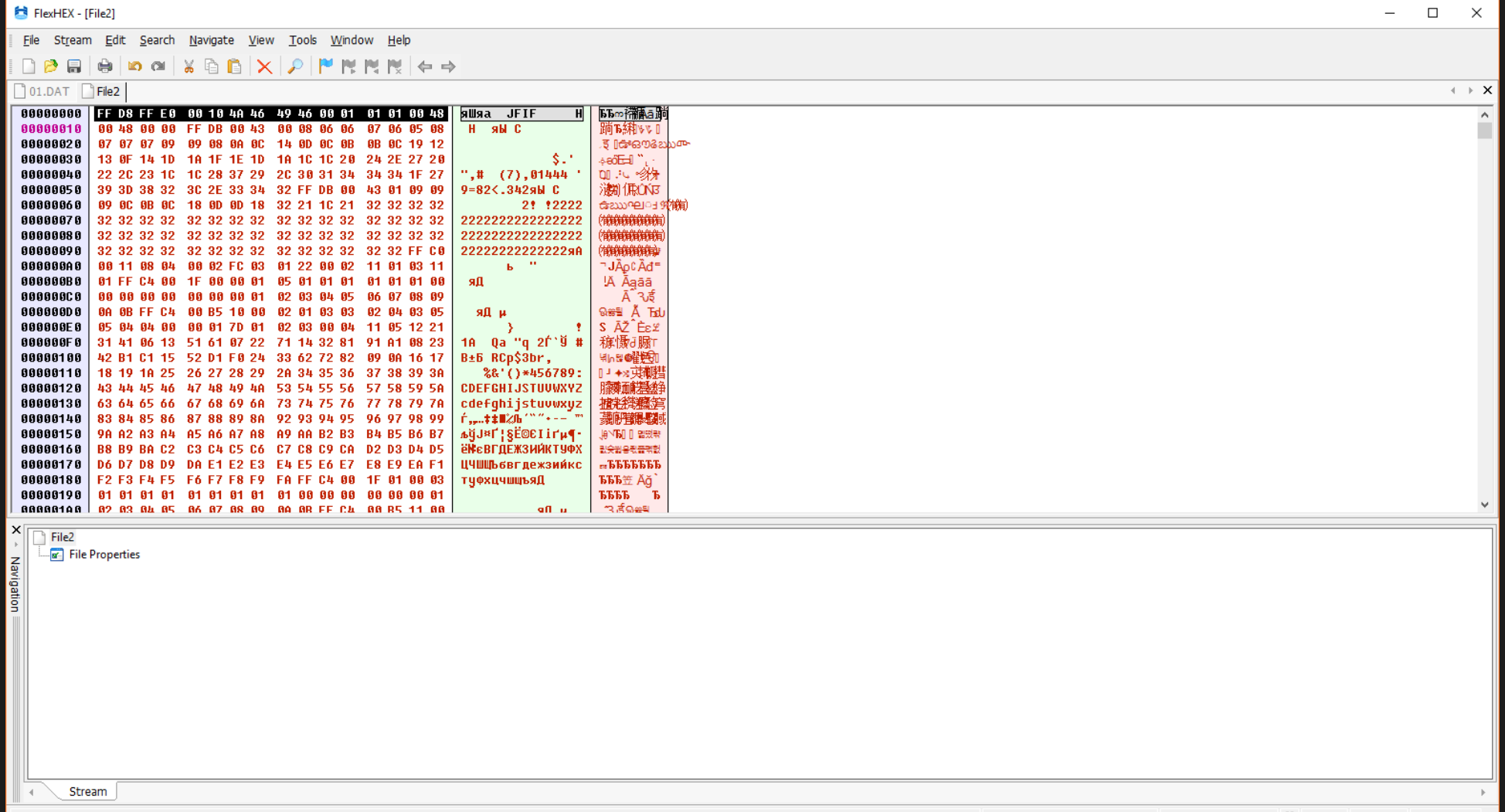
В нижнее поле введем **32** в качестве проверки того, что мы получим на выходе.**Это и будет нашим вторым операндом.**


Bitwise Operation
Operation: XOR (Exclusive OR)
Second Operand: Hex Bytes
32

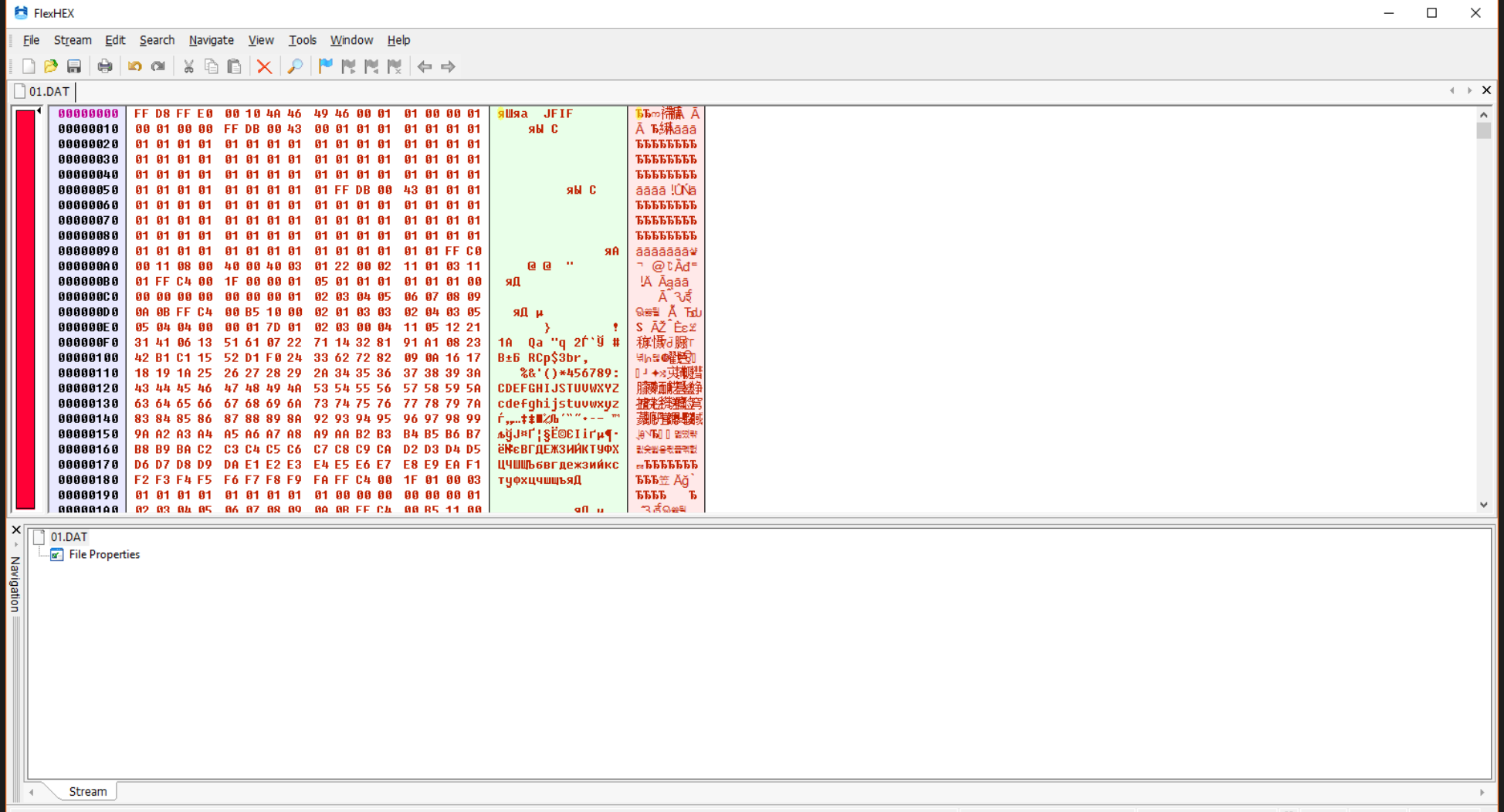
Поксорили...



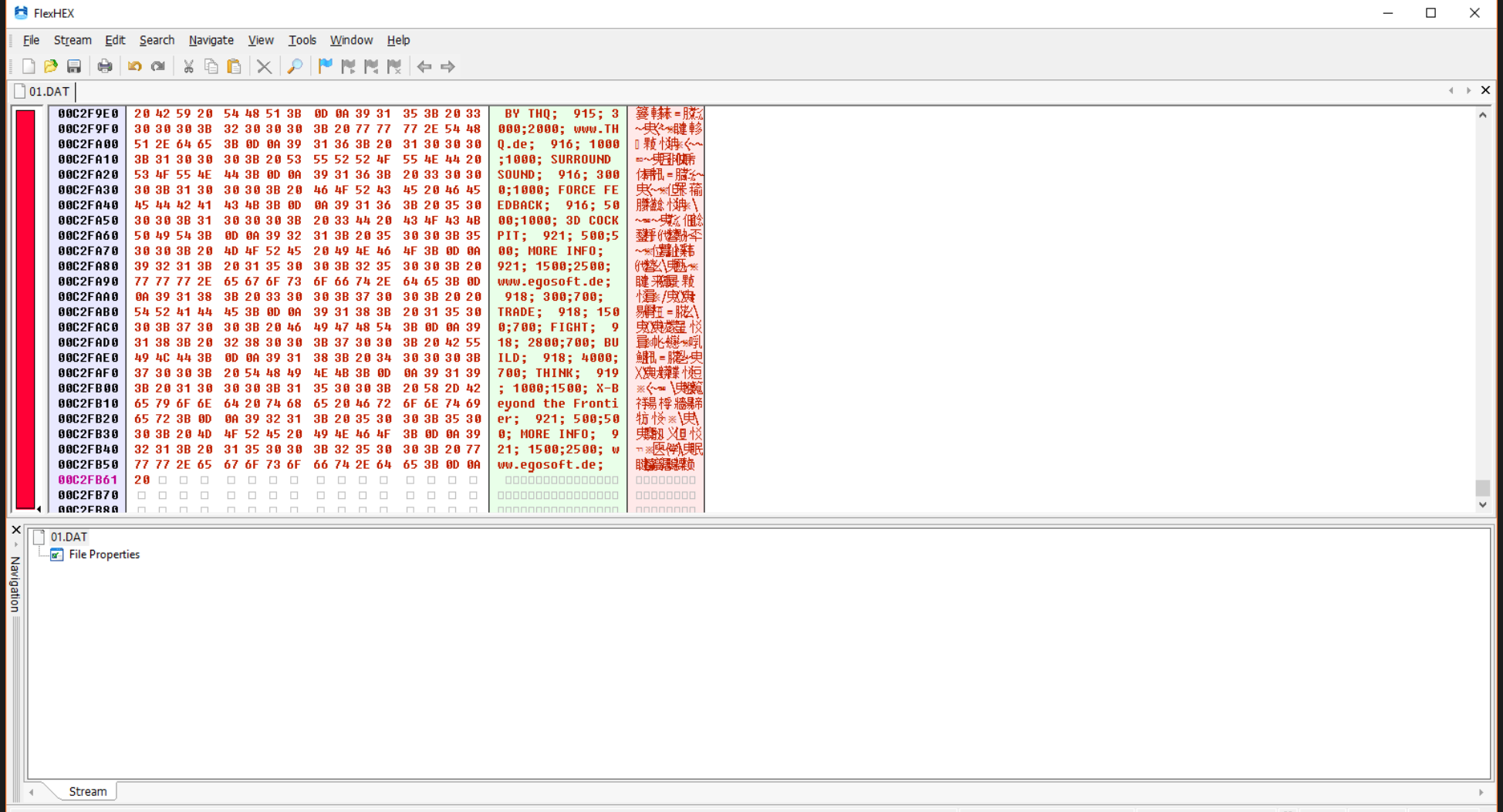
Хм.Ничего интересного, хотя.Что-то мне это напоминает.ДА! Это очень похоже на заголовок **jpg** изображений.



Значит это действительно **XOR** и на этот раз вторым операндом у нас будет **33**.
Жмем **Ctrl+Z** для отмены предыдущей операции и выполняем **XOR** повторно с новым операндом.
Получаем на выходе следующее:



Пролистаем файл в самый низ, чтобы убедиться в том, что он полностью расшифрован.



Все ок.Но возникла проблема номер два.Как определить границы файлов ?
Рядом с вышеописанным файлом лежит **01.CAT**, который, судя по его названию, представляет из себя специфичный заголовок и **TOC**, а из-за своего малого размера вполне подходит для хранения данных о файловой таблице имен и смещений.
И он тоже зашифрован, но уже по другому алгоритму.В детали поиска я вдаваться не буду, могу только дополнить, что код расшифровки выполняется после вызова стандартных функций для работы с файлами, а именно в этом месте:
==

```
00472A13   > /8A62 01       MOV AH,BYTE PTR DS:[EDX+1]
00472A16   . |8AF8          MOV BH,AL
00472A18   . |FEC7          INC BH
00472A1A   . |8A1A          MOV BL,BYTE PTR DS:[EDX]
00472A1C   . |32C3          XOR AL,BL
00472A1E   . |8ADF          MOV BL,BH
00472A20   . |8802          MOV BYTE PTR DS:[EDX],AL
00472A22   . |32DC          XOR BL,AH
00472A24   . |8AC7          MOV AL,BH
00472A26   . |FEC0          INC AL
00472A28   . |885A 01       MOV BYTE PTR DS:[EDX+1],BL
00472A2B   . |8B5D F0       MOV EBX,DWORD PTR SS:[EBP-10]
00472A2E   . |83C2 02       ADD EDX,2
00472A31   . |3BD3          CMP EDX,EBX
00472A33   . |8B5D F8       MOV EBX,DWORD PTR SS:[EBP-8]
00472A36   .^\7C DB         JL SHORT x.00472A13
```

==

Цикл расшифровки в x.exe.

Краткая суть алгоритма:

1. Берем первый байт ;
2. Берем ключ расшифровки = **DBh** ;
3. Ксорим его с первым байтом ;
4. Увеличиваем (получаем **DCh)**;
5. Ксорим второй байт с увеличенным значением ;
6. Снова увеличиваем (получаем **DDh**);
7. Повторяем с первого шага, переходя к следующей паре байт.

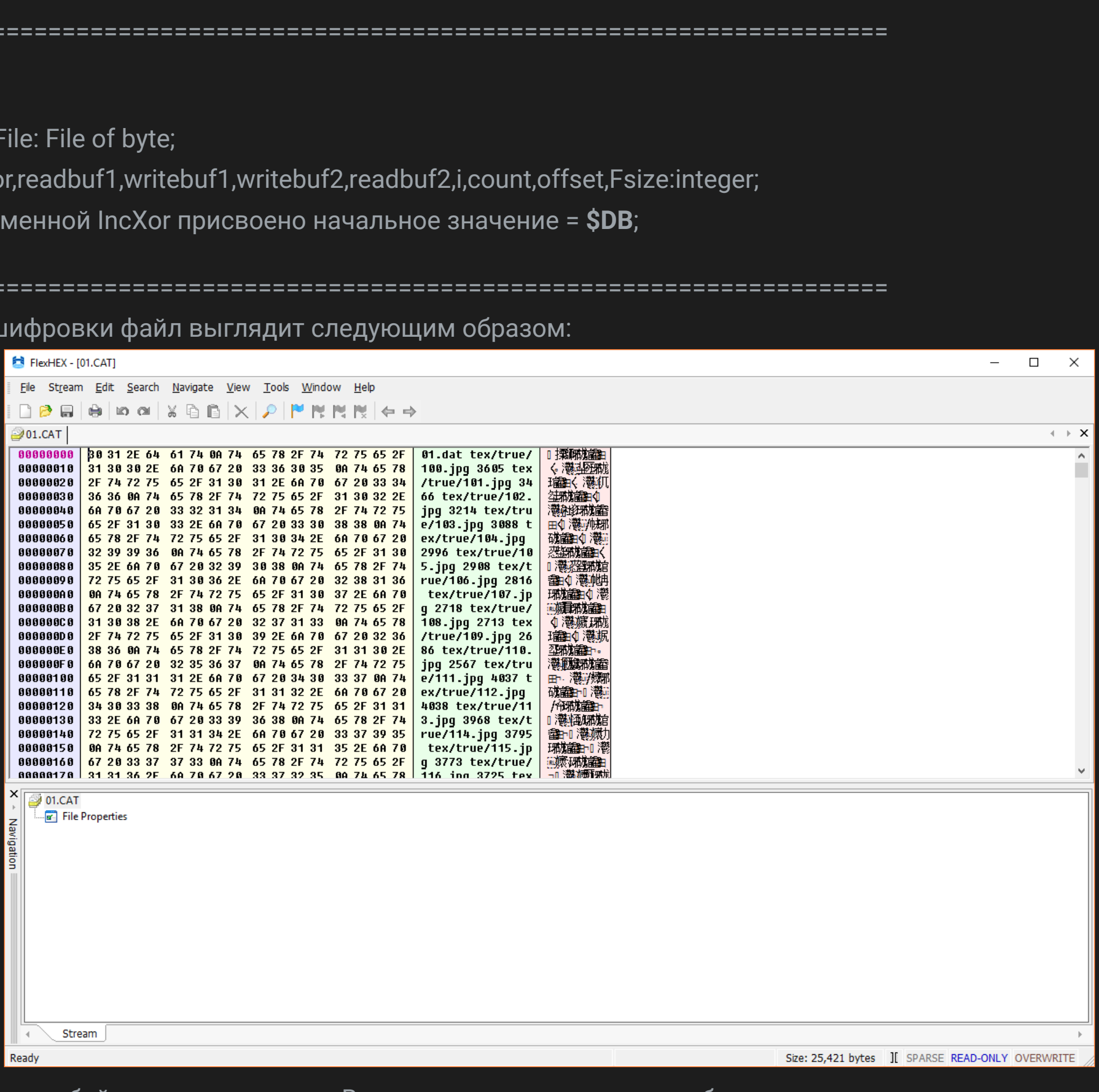
На делфи получился бы примерно такой код цикла:
==

```
for I := 0 to Fsize+3 - 1 do  //прибавим три байта, чтобы выравнять размер для кратности двум
    begin
      try
        Application.ProcessMessages;
        //Для каждого первого байта
        BlockRead(DataFile,ReadBuf1,1);
        WriteBuf1:=ReadBuf1 xor IncXor;
        Seek(datafile,offset);
        BlockWrite(Datafile,Writebuf1,1);
        inc(count);
        Label2.Caption:=IntToStr(count);
        ProgressBar1.Position:=ProgressBar1.Position+1;
        inc(offset);
        IncXor:=IncXor+1;
       //Для каждого второго байта
        BlockRead(DataFile,ReadBuf2,1);
        WriteBuf2:=IncXor xor ReadBuf2;
        Seek(datafile,offset);
        BlockWrite(Datafile,Writebuf2,1);
        inc(count);
        Label2.Caption:=IntToStr(count);
        ProgressBar1.Position:=ProgressBar1.Position+1;
        inc(offset);
        IncXor:=IncXor+1;
      except
      end;
    end;
```

==
Где:

1. DataFile: File of byte;
2. IncXor,readbuf1,writebuf1,writebuf2,readbuf2,i,count,offset,Fsize:integer;
3. Переменной IncXor присвоено начальное значение = **$DB**;

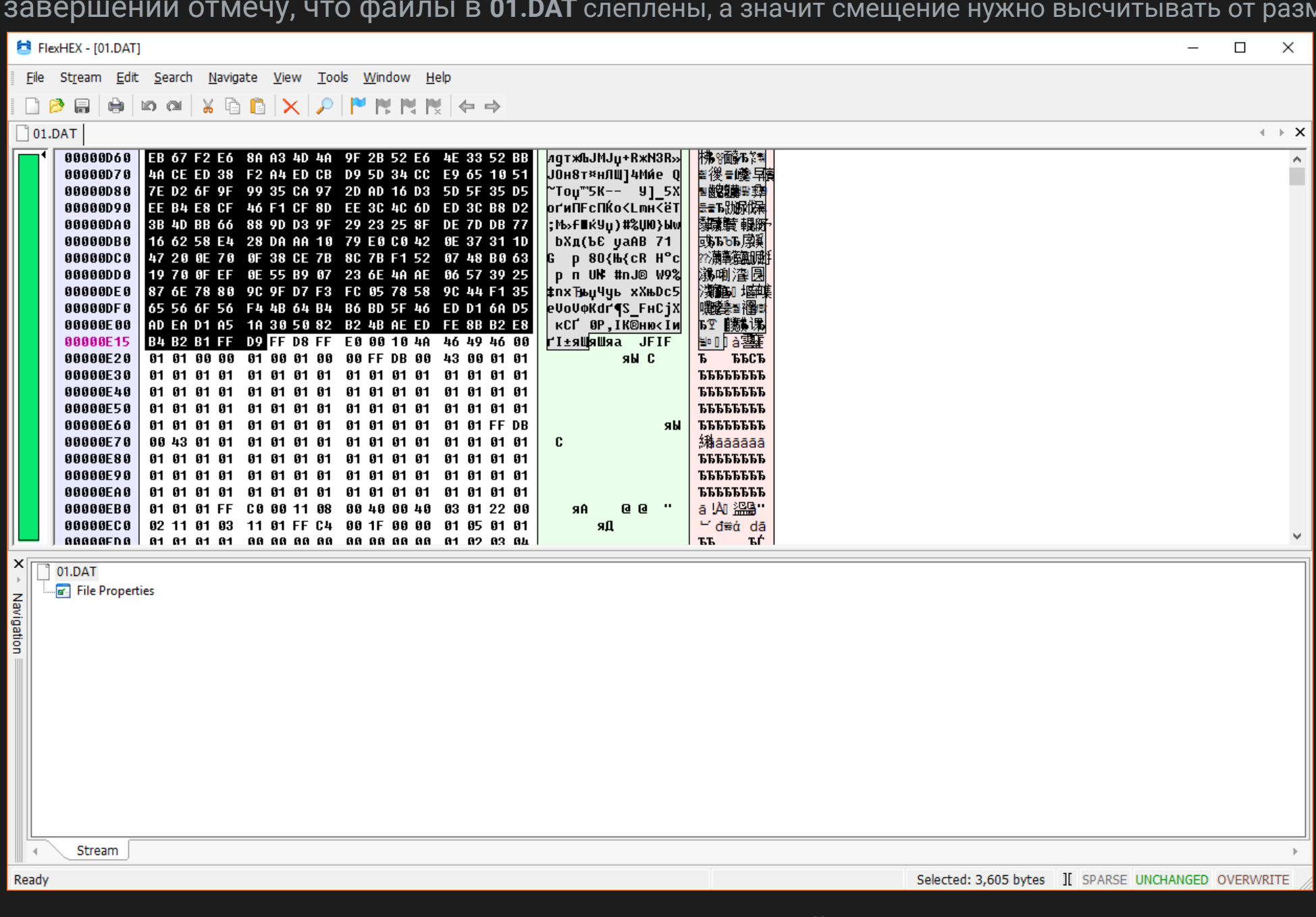
==
После расшифровки файл выглядит следующим образом:



Первые шесть байт - внутреннее имя.Вполне вероятно, что таким образом проверяется валидность.Седьмой байт - символ завершения строки.

Далее отчетливо видны имена файлов с полными путями и их размеры, представленными в виде строк.

В завершении отмечу, что файлы в **01.DAT** слеплены, а значит смещение нужно высчитывать от размера предыдущего.



На этом мутная хуйня закончилась.