Is Black Block a black box? (Spoiler: No)

Augustin Bariant (graniter)

May 2, 2023

1 Challenge description

The challenge gives us a sage script implementing a non traditionnal stream cipher. The cipher uses a 128-bit secret key K and generates an IV at each encryption. This IV is interpreted as counter c: for each block of 128 bits of plaintext, the encryption function takes the key stream generated from K and c, and XORs it to the plaintext. The counter c is then incremented for the next plaintext block. Basically, we have (+ here denotes the XOR):

$$C_i = P_i + E_k(IV + i)$$

Instead of manipulating bits like traditionnal ciphers, the function E_k converts the 128-bit counter and the 128-bit key into elements of $GF(2^{128})$ and performs operations in the big field. In cryptography, this class of algorithm is refered as **arithmetization-oriented**.

Together with this implementation, we also get a text file containing the encryption of a 48-byte known plaintext, under a known IV. The text file also contains the encryted flag under a known IV.

This challenge was designed by Henri Gilbert and was solved by 3 persons.

2 Analysis of the challenge

Let us look inside the black block: the keystream generation function E.

```
# K is the key as an element of GF(2^{128})
# B is the counter as an element of GF(2^{128})
# a is an element of GF(2^{128})
def E(self, B):
    K = self._K
    for _ in range(64):
        B += K
        if B != 0:
            B = (1 + a + a**3) / B**4
        K = (K + a)**3
    B += K
    return B
```

Let's try to understand the polynomial (or rather ractionnal function) $B, K \to E(B, K)$, with different numbers of rounds.

```
from sage.all import *
FF = GF(2 ** 128,names = ('a'))
a = FF.gen()
Pol_field = PolynomialRing(FF, names=('c', 'k'))
(c, k) = Pol_field._first_ngens(2)
F = Pol_field.fraction_field()
def E(B,r):
    K = k
    for i in range(r):
        B += K
        if B != 0:
            B = (1 + a + a**3) / B**4
        K = (K + a)**3
        print(i)
    B += K
    return B
print(E(c,1))
#(c^4*k^3 + k^7 + (a)*c^4*k^2 + (a)*k^6 + (a^2)*c^4*k + (a^2)*k^5 + (a^3)*c^4 +
↪  (a^3)*k^4 + (a^3 + a + 1))/(c^4 + k^4)
print(E(c,2))
#(c^16*k^21 + k^37 + (a)*c^16*k^20 + (a)*k^36 + (a)*c^16*k^18 + (a)*k^34 +
↪  (a^4)*c^16*k^17 + (a^4)*k^33 + (a^5 + a^3)*c^16*k^16 + (a^5 + a^3)*k^32 +
↪  (a^2)*c^16*k^15 + (a^2)*k^31 + (a^3)*c^16*k^14 + (a^3)*k^30 + (a^4)*c^16*k^13
↪  + (a^4)*k^29 + (a^5 + a^3)*c^16*k^12 + (a^5 + a^3)*k^28 + (a^6)*c^16*k^11 +
↪  (a^6)*k^27 + (a^7)*c^16*k^10 + (a^7)*k^26 + (a^8)*c^16*k^9 + (a^8)*k^25 + (a^9
↪  + a^7)*c^16*k^8 + (a^9 + a^7)*k^24 + (a^10)*c^16*k^7 + (a^10)*k^23 +
↪  (a^11)*c^16*k^6 + (a^11)*k^22 + (a^16 + a^12)*c^16*k^5 + (a^16 + a^12)*k^21 +
↪  (a^17 + a^13 + a^11)*c^16*k^4 + (a^17 + a^13 + a^11)*k^20 + (a^14)*c^16*k^3 +
↪  (a^14)*k^19 + (a^17 + a^15)*c^16*k^2 + (a^17 + a^15)*k^18 + (a^20 +
↪  a^16)*c^16*k + (a^20 + a^16)*k^17 + (a^21 + a^19 + a^17 + a^15 + a^3 + a +
↪  1)*c^16 + (a^21 + a^19 + a^17 + a^15 + a^3 + a + 1)*k^16 + (a^12 + a^4 +
↪  1)*k^9 + (a^13 + a^5 + a)*k^8 + (a^13 + a^5 + a)*k^6 + (a^15 + a^7 + a^3)*k^4
↪  + (a^14 + a^6 + a^2)*k^3 + (a^17 + a^15 + a^9 + a^7 + a^5 + a^3)*k^2 + (a^20 +
↪  a^16 + a^12 + a^4)*k + (a^21 + a^19 + a^17 + a^15 + a^13 + a^11 + a^5 +
↪  a^3))/(c^16*k^12 + k^28 + (a^4)*c^16*k^8 + (a^4)*k^24 + (a^8)*c^16*k^4 +
↪  (a^8)*k^20 + (a^12)*c^16 + (a^12)*k^16 + (a^12 + a^4 + 1))
```

For a good round function, we would expect a random-looking rationnal function. But here, c only appears under the form c^4 (for $r = 1$) and c^{16} (for $r = 2$). Small experiments showed that for $r = i$ ($i \leq 4$), c only appears under the form c^{4^i}. This can actually be proven by induction, and comes from the fact that $x \to x^4$ is linear in $GF(2^{128})$ (proof given in appendix **??**). We therefore have E_k for the following form:

$$E_k(c, r) = \frac{P_r(k) + Q_r(k) \times x^{4^r}}{R_r(k) + S_r(k) \times x^{4^r}}$$

where P_r, Q_r, R_r and S_r are (r-dependant) polynomials. This property is used to mount an attack in the next section.

3 Solving the challenge

Recall that the total number of rounds is 64. Therefore, the full function $E_k(c)$ is of the form

$$E_k(c) = \frac{P(k) + Q(k) \times c^{4^{64}}}{R(k) + S(k) \times c^{4^{64}}}$$

We now only have to spot that $c^{4^{64}} = c^{2^{128}} = c$ in $GF(2^{128})$ since the group $(GF(2^{128})^*, \times)$ is of cardinal $2^{128} - 1$. This implies that:

$$E_k(c) = \frac{P(k) + Q(k) \times c}{R(k) + S(k) \times c}$$

Since k is fixed in our case, we will omit the k and can actually rewrite $E_k(c)$ as:

$$E(c) = \frac{p + c}{r + sc}$$

with unknown (key-dependant) p, r, s. With the 3 128-bit pairs of plaintexts and ciphertexts, we can retrieve the keystream by XORing each plaintext with its corresponding ciphertext, yielding 3 128-bit keystream output under three consecutive counters (the addition here is performed in $\mathbb{Z}_{2^{128}}$): $(c, c+1, c+2)$. In the settings of the challenge, we verify that the counter additions are also true in $GF(2^{128})$, where the addition is in fact a XOR. This is not true in general (for instance, $2 = 1 + 1$ in $\mathbb{Z}_{2^{128}}$ but $2 = 1 + 3$ after conversion in $GF(2^{128})$). With the knowledge of this key stream, we derive three equations in $GF(2^{128})$:

$$\begin{cases} E(c) = \dfrac{p + c}{r + sc} \\ E(c+1) = \dfrac{p + c + 1}{r + sc + s} \\ E(c+2) = \dfrac{p + c + 2}{r + sc + 2s} \end{cases}$$

or (because $+$ is $-$ in $GF(2^{128})$)

$$\begin{cases} p + E(c+0)r & +E(c+0)(c+0)s = & c \\ p + E(c+1)r & +E(c+1)(c+1)s = & c+1 \\ p + E(c+2)r & +E(c+2)(c+2)s = & c+2 \end{cases}$$

This gives 3 linear equations with 3 unkown variables (p, r, s). We solve this linear system to get (p, r, s) and we fully know the mapping $c \to E(c)$. Then, all we have to do is take the counters c_i corresponding to the IV of the encrypted flag, compute $E(c_i)$ and decrypt the encrypted flag.

```python
import json
def byte_xor(ba1, ba2):
    return bytes([_a ^ _b for _a, _b in zip(ba1, ba2)])

dic = {}
with open("output.txt","r") as f:
    dic_str = f.readline()
    dic = json.loads(dic_str)
    flag_dic_str = f.readline()
    flag_dic = json.loads(flag_dic_str)
iv = bytes.fromhex(dic['iv'])
p = bytes.fromhex(dic['p'])
c = bytes.fromhex(dic['c'])
flag_iv = bytes.fromhex(flag_dic['iv'])
flag_enc = bytes.fromhex(flag_dic['flag_enc'])
key_stream  = byte_xor(p,c)
cnt = int.from_bytes(iv, 'big')
flag_cnt = int.from_bytes(flag_iv, 'big')
E_known = {cnt: int.from_bytes(key_stream[0:16],'big'), cnt + 1:
↪  int.from_bytes(key_stream[16:32],'big'), cnt + 2:
↪  int.from_bytes(key_stream[32:48],'big')}


Eq0 = [FF(1), FF.fetch_int(E_known[cnt+0]),
↪  FF.fetch_int(E_known[cnt+0])*FF.fetch_int(cnt+0)]
Eq1 = [FF(1), FF.fetch_int(E_known[cnt+1]),
↪  FF.fetch_int(E_known[cnt+1])*FF.fetch_int(cnt+1)]
Eq2 = [FF(1), FF.fetch_int(E_known[cnt+2]),
↪  FF.fetch_int(E_known[cnt+2])*FF.fetch_int(cnt+2)]
M = Matrix([Eq0,Eq1,Eq2])
Y_mat = Matrix([FF.fetch_int(cnt + 0),FF.fetch_int(cnt + 1),FF.fetch_int(cnt +
↪  2)])
Sol = M.inverse()*Y_mat.transpose()


p = Sol[0][0]
r = Sol[1][0]
s = Sol[2][0]

flag_key_stream = []
curr_cnt = flag_cnt
for i in range(0,len(flag_enc),16):
    curr_key_stream = (p + FF.fetch_int(curr_cnt)) / (r +
    ↪  FF.fetch_int(curr_cnt)*s)
    curr_key_stream =
    ↪  int.to_bytes(curr_key_stream.integer_representation(),16,byteorder="big")
    for j in range(16):
        flag_key_stream.append(curr_key_stream[j])
    curr_cnt += 1
flag_key_stream = bytes(flag_key_stream)

print(byte_xor(flag_enc,flag_key_stream))
```

A Proof by induction

Let us call $c \to E_k(c, r)$ the rational function corresponding to E with r rounds and with the key k. Let us prove by induction that for all $r \geq 0$, the mapping $x \to E_k(x, r)$ is of the form

$$\frac{P_r(k) + Q_r(k) \times x^{4^r}}{R_r(k) + S_r(k) \times x^{4^r}}$$

where P_r, Q_r, R_r, S_r are polynomials. For readability, we will omit the index r. Let us suppose that this is true for r. Let us also denote $T_r(k)$ the r-th round key (K).

$$\begin{aligned} E_k(c, r+1) &= \frac{\alpha}{E_k(c, r)^4} + T_{r+1}(k) \\ &= \frac{\alpha(R(k) + S(k)x^{4^r})^4}{(P(k) + Q(k)x^{4^r})^4} + T_{r+1}(k) \end{aligned}$$

Since $x \to x^4$ is linear in $GF(2^{128})$, $(a+b)^4 = a^4 + b^4$. This can be proven using the field characteristic 2.

$$\begin{aligned} E_k(c, r+1) &= \frac{\alpha(R(k) + S(k)x^{4^r})^4}{(P(k) + Q(k)x^{4^r})^4} + T_{r+1}(k) \\ &= \frac{\alpha R(k)^4 + \alpha S(k)^4 x^{4^{r+1}}}{P(k)^4 + Q(k)^4 x^{4^{r+1}}} + T_{r+1}(k) \\ &= \frac{\alpha R(k)^4 + \alpha S(k)^4 x^{4^{r+1}} + T_{r+1}(k)(P(k)^4 + Q(k)^4 x^{4^{r+1}})}{P(k)^4 + Q(k)^4 \times x^{4^{r+1}}} \end{aligned}$$

And so it is the case for $r+1$. Since it is trivially true for $r = 0$, the property is proven.