Cryptanalysis of F.E.A.L.

BERT DEN BOER

Centre for mathematics and computescience (*)
Kruislaan 413
1098 SJ AMSTERDAM, The NETHERLANDS

Summary

At Eurocrypt 87 the blockcipher F.E.A.L. was presented [2]. Earlier algorithms called F.E.A.L.-1 and F.E.A.L.-2 had been submitted to standardization organizations but this was presumably the final version. It is a Feistel cipher, but in contrast to D.E.S., a software implementation does not require a table look-up. The intention was a fast software implementation and also an avoidance of discussions about random tables. As Walter Fumy indicated at Crypto 87 [1] a certain transformation on 32 bits used by the cipher was not complete in contrast to a remark made during the presentation of F.E.A.L. at Eurocrypt 87. Furthermore, the transformation is too close to a quadratic function on the input.

I am informed that after my informal expose at Crypto 87 about certain vulnerabilities of F.E.A.L., its designers have created F.E.A.L.-8 with twice as many rounds. Later on again versions were renamed. The (definite?) version in the abstracts [2] without a serial number got version number 1.00 and F.E.A.L.-8 got version number 2.00 in the proceedings of Eurocrypt '87 [3]. In this paper we shall show that F.E.A.L. as presented at Eurocrypt 87 is vulnerable for a chosen plaintext attack which requires at most ten thousand plaintexts.

Encryption Algorithm

For convenience and definiteness we first reformulate the encipherment algorithm. The F.E.A.L.-algorithm is a blockcipher acting on 64 bits of plaintext to produce a 64 bit ciphertext controlled by a 64 bit key.

One of the buildingblocks of the cipher is a transformation $S$ from $F_2^8 \times F_2^8 \rightarrow F_2^8$ defined by

$$S(x,y,a) = \text{Rot}((x+y+a) \mod 256)$$

*This research was supported by the Netherlands Organization for Advancement of Pure Research

Copyright (c) 1998, Springer-Verlag
i.e. the 8 bit numbers \( x \) and \( y \) are considered as residues mod 256, \( a \) is the residue class of 0 or 1 and Rot cyclically rotates the bits of its input 2 places such that the 6 least significant bits become the 6 most significant bits. Another building block is the exclusive-or on two bytes denoted by \( \oplus \). The same notation will be used for the exclusive-or sums of four byte strings. We define a \( f_k \)-box as follows: \( f_k \) transforms 2 strings of 4 bytes \( L \) and \( R \) into a four byte string \( O \) as follows: (In shorthand \( f_k(L,R)=O \))

denote the input by \( L(0) \) up to \( L(3) \) and \( R(0) \) up to \( R(3) \) and the output by \( O(0) \) up to \( O(3) \) then:

\[
\begin{align*}
\text{hulp}&=L(2) \oplus L(3) \\
O(1)&=S((L(0) \oplus L(1)),(\text{hulp} \oplus R(0)),1) \\
O(0)&=S(L(0),(O(1) \oplus R(2)),0) \\
O(2)&=S(O(1) \oplus R(1),\text{hulp},0) \\
O(3)&=S((O(2) \oplus R(3)),L(3),1)
\end{align*}
\]

The function \( G \) transforms one string of four bytes into one string of four bytes as follows: (In shorthand \( G(I)=O \)) denote the input by \( I(0) \) up to \( I(3) \) and the output by \( O(1) \) up to \( O(3) \), then:

\[
\begin{align*}
\text{hulp}&=I(2) \oplus I(3) \\
O(1)&=S(I(0) \oplus I(1),\text{hulp},1) \\
O(2)&=S(O(1),\text{hulp},0) \\
O(3)&=S(O(2),I(3),1) \\
O(0)&=S(O(1),O(0),0)
\end{align*}
\]

The blockcipher consists of a key schedule and a data randomizer. The keyschedule operates as follows: The eight byte input is considered as two strings \( A_0 \) and \( B_0 \) of four bytes each. Further a four byte string \( C_0 \) with all 32 bits zero is introduced. Iteratively \( A_i, B_i, C_i, i=1, \ldots, 6 \) are defined by

\[
\begin{align*}
B_{i+1} &= f_k(A_i; (C_i \oplus B_i)) \\
C_{i+1} &= A_i \\
A_{i+1} &= B_i
\end{align*}
\]

Further we need two simple functions \( PL \) and \( PR \) transforming four byte strings as follows:
PL(u,v,w,x) = (0,u,v,0)
PR(u,v,w,x) = (0,w,x,0).

The strings \( B_1, \ldots, B_6 \) of the keyschedule are transformed into 6 strings \( M_i \), \( i=0,\ldots,5 \) as follows:

\[
\begin{align*}
M_0 &= B_3 \oplus PR(B_1) \\
M_1 &= B_3 \oplus B_4 \oplus PL(B_1) \\
M_2 &= PL(B_1) \oplus PL(B_2) \\
M_3 &= PR(B_1) \oplus PR(B_2) \\
M_4 &= B_5 \oplus B_6 \oplus PR(B_1) \\
M_5 &= B_5 \oplus PL(B_1).
\end{align*}
\]

The datarandomizer operates as follows (see fig 2): The 64 bit input is viewed as two strings \( P_0 \) and \( P_1 \) of four bytes. Now we define

\[
\begin{align*}
D_0 &= P_0 \oplus M_0 \\
E_0 &= P_0 \oplus P_1 \oplus M_1 \\
D_1 &= E_0 \\
E_1 &= D_0 \oplus G(E_0) \\
D_2 &= E_1 \\
E_2 &= D_1 \oplus G(E_1) \\
D_3 &= E_2 \\
E_3 &= D_2 \oplus G(E_2 \oplus M_2) \\
D_4 &= D_3 \oplus G(E_3 \oplus M_3) \oplus M_5 \\
E_4 &= E_3 \oplus M_4 \\
C_0 &= D_4 \\
C_1 &= D_4 \oplus E_4
\end{align*}
\]

Finally the two strings \( C_0 \) and \( C_1 \) of four bytes each are concatenated to form the 64-bit ciphertext.

**Cryptanalysis**

To determine the key we use a chosen plaintext attack. The choice of the plaintext depends on results derived from previous plaintext and ciphertext. We are going to determine the 160 unknown bits in the \( M_i \)'s as though there is no relation between them. Once they are determined we can decipher any ciphertext but we also can use the keyschedule from the bottom to determine the 64-bit
key. This process will not require more than ten thousand plaintexts.

Observe the value $C_0 \oplus C_1$. It is equal to

$$P_0 \oplus M_4 \oplus M_0 \oplus G(E_0) \oplus G(E_0) \oplus M_2 \oplus G(G(E_0) \oplus M_0 \oplus P_0).$$

Assume that $P_0 \oplus P_1$ is a constant, then $E_0$ and $G(E_0)$ are constants too. Define

$$K_0 = G(E_0) \oplus M_0$$
$$K_1 = E_0 \oplus M_2$$
$$K_2 = M_4 \oplus M_0 \oplus G(E_0).$$

$$CP = C_0 \oplus C_1 \oplus P_0$$

then:

(1) \hspace{1cm} CP = K_2 \oplus G(K_1) \oplus G(K_0 \oplus P_0).

Formula (1) is the crucial formula. By keeping the exclusive-or sum of $P_0$ and $P_1$ constant it is possible to determine the constants $K_0$ up to $K_2$ with at most say 300 choices of $P_0$.

Define

$$K_0 = (x^0, x^1, x^2, x^3)$$
$$K_1 = (y^0, y^1, x^2, x^3)$$
$$K_2 = (z^0, z^1, z^2, z^3)$$
$$P_0 = (a^0, a^1, a^2, a^3)$$
$$CP = (f^0, f^1, f^2, f^3).$$

See figure 1 where internal bytes $b^k, c^k, d^k, e^k$ are defined within the picture.

The idea is to solve $K_0$ first. The first bits to solve are the 6 least significant bits of $x^0$. This starts by keeping $a^3, a^2, a^1 \oplus a^0$ constant and also the two most significant bits $a^0$ and study the behaviour of one particular bit $f^{15}$ for the remaining 64 cases. Observe that $b^1, b^2, b^3, c^1, c^2, c^3, d^2, d^3$ are constant in those cases. Let $b^{01} = b^0 \mod 64$ and $c^{11} = c^1 \mod 64$ and carry $= (b^{01} + c^{11}) \div 64$. Then it holds for the bits $c^{01}, c^{01}, c^1, c^1, c^1, c^1, c^1, f^{15}$ that their value is of the form "constant 7 carry". The value $c^{11}$ is a constant and as the 6 least significant bits of $a^0$ assume all 64 possibilities and so $b^{01}$ assumes all 64 possible values. Counting the number of times $f^{15}$ is equal to one, leaves us with at most two possibilities for $c^{11}$.

In order to determine which possibility holds for $c^{11}$ observe that changing $a^1$ or $a^0$ the six most significant bits of $c^1$ and therefore the four most significant bits of $c^{11}$ remain constant. Combining the results of two or three

$M_1$ except for a freedom of two bits.

Finally we study the values $C_0$ we have encountered up to this moment. Those give equations of the form

$$Q_1 = M_5 \oplus G(M_3 \oplus Q_2)$$

where $Q_1$ and $Q_2$ are known. Considering the fact that up to now we have between
counts will give only one consistent possibility for the two or three values of \( c^{11} \). The actual counting never requires the full 192 ciphertexts but at most 127 ciphertexts in special cases (in a very favourable case 10 is enough).

To determine the 6 least significant bits of \( x^0 \) note that at least one of the two or three actual values of \( c^{11} \) is odd. In that case there exist exactly one value \( b^{01} \) such that \( b^{01} \) will give carry=1 and \( b^{01} \oplus 1 \) will give carry=0. From this we conclude that \( b^{01} \) equals 64-\( c^{11} \). We know the corresponding value of \( a^0 \) so indeed we can determine the six least significant bits of \( x^0 \).

To proceed we use this knowledge and start changing the lowest bit of \( a^0 \oplus a^1 \). Two well-chosen plaintexts and the corresponding values of \( f^{15} \) is enough to determine the least significant bit of \( x^0 \oplus x^1 \). The same is true for the next two bits of \( x^0 \oplus x^1 \). Simultaneously the three least significant bits of \( x^2 \oplus x^3 \) are determined. To determine the next three bits of \( x^0 \oplus x^1 \) and \( x^2 \oplus x^3 \) might require 42 plaintexts in the worst case. Still only the value of \( f^{15} \) is all what we need of the ciphertext.

Along similar lines we can determine \( x^0 \oplus x^1 \) and \( x^2 \oplus x^3 \), the seven least significant bits of \( x^0 \) and the seven least significant bits of \( x^3 \). For the moment we are allowed to assume that \( x^0_0 \) and \( x^3_0 \) are zero. In other words \( K_0 \) is determined and at the cost of at most 250 plaintexts.

Once \( K_0 \) is determined the determination of \( K_1 \) and \( K_2 \) is easy and will cost at most 30 well chosen plaintexts with the corresponding ciphertexts. There is a freedom in \( K_1 \) of two bits but we can just do a choice.

Now observe what happens if we change \( P_0 \oplus P_1 \). Then the new value of \( K_1 \) is known. With the above described technique we establish the new value of \( K_0 \). Then \( K_2 \) follows directly because of a linear relation.

This results in knowledge of \( M_0 \oplus G(M_1 \oplus (P_0 \oplus P_1)) \) for values \( P_0 \oplus P_1 \) of our own choosing. With say at most 30 values we can establish \( M_0 \) and \( M_1 \) except for a freedom of two bits.

Finally we study the values \( C_0 \) we have encountered up to this moment. Those give equations of the form

\[ Q_1 = M_5 \oplus G(M_3 \oplus Q_2) \]

where \( Q_1 \) and \( Q_2 \) are known. Considering the fact that up to now we have between...
100 and 10000 ciphertexts it is safe to assume that we have enough data to determine $M_3$ and $M_5$.

Combining this knowledge we can decipher any ciphertext. If we want to recover the original key we use the restricted possibilities for $M_2$ and $M_3$ to reduce the uncertainty in $M_0$ up to $M_5$. Given those $M_i$’s we can use these data and the last $f_k$-box to solve $B_5$ and $B_4$ and a few more bytes. After that we can simply try the 256 possibilities for $B_3(2)$ and resolve the keyschedule.

Conclusions

In the presented version the $G$-box is too regular. If one wants this small number of rounds(4) a better design should be possible. In [3] the algorithm with twice as many rounds is considered by the authors to be secure because four statistical values are close or equal to theoretical values but the same argument was used for the algorithm presented at Eurocrypt ’87. As this turned out not to be sufficient one should use other arguments for the security of an encipherment algorithm.

Acknowledgement

The author wishes to thank D. Chaum and W. Fumy for a challenging remark which made me start the investigations. Further the author wishes to thank D. Chaum for stimulation during the investigations. The author also wishes to thank T. Siegenthaler for remarks on a draft version of this article.

References
