The DFC Cipher : an attack on careless implementations

Ian Harvey, nCipher Corporation Ltd

Abstract: The Decorrelated Fast Cipher (DFC) presented as a candidate for the Advanced Encryption Standard (AES), contains a primitive operation which, in typical implementations, is susceptible to an attack which can recover some key material. Whilst this is not a mathematical flaw in the algorithm, it presents concerns that a correct implementation may be difficult to achieve.

1. Description of DFC

DFC [1] is a 128-bit block cipher proposed for the future Advanced Encryption Standard to replace DES. It is conventional Feistel-structure cipher, using a 64-bit round function, over 8 rounds.

The round function RF, takes a 64-bit input *x*, two 64-bit parameters *a* and *b* derived from the key, and computes:

RF(x,a,b) = CP(($ax+b$) mod (W+13) mod W)

where W is the value 2^{64} and CP (the 'Confusion Permutation') is a fixed function. The precise details of CP are not relevant.

2. Computation of RF

The core of the round function, requiring the majority of the computational effort, is the calculation of $ax+b$ mod (W+13). a, x, and b are 64-bit quantities, so $ax+b$ has up to 128 bits. Reducing this modulo a 65-bit number (W+13) is beyond the range of most processors' instructions, and a full long division would require an impractical amount of CPU cycles, so an iterative approach similar to the following is almost always taken.

In the following description, A to F are 64-bit working variables, W is the quantity 2^{64} and the notation A::B means A concatenated to B (i.e. the value WA+B).

Input: *a, x* and *b*
Output: ($ax+b$) mod (W+13) mod W

1. A::B = (a*x + b)
   // Range 0..FFFFFFFF FFFFFFFF 00000000 00000000
2. C::D = (A*13)

```
        // D has range 0..W-1, C has range 0..12
3.   E::F = (signed) B + 13*C - D
        // Range -(W-1) to (W-1)+156
        // Here we have calculated
        //  B + 13C - D =
        //  (WA+B)-WA-(WC+D) + (WC+13C) =
        //  (WA+B) - WA -13A + (W+13)C =
        // congruent to (WA+B) mod (W+13)
4.   If E::F < 0
        return (F+13) mod W
        // Add on (W+13), give answer mod W
        // range 0..W-1
5.   If E::F < W+13
        return (F)
        // Implicit mod W
        // range 0..W-1
6.   Otherwise,
        return (F-13)
        // Take off (W+13)
        // Range 0..142
```

The important thing to notice in this algorithm (and many similar variants) is that the execution path is variable and more particularly, *for some execution paths, the range of possible outputs is limited.* In particular, in this algorithm step 6 can only produce a fraction of the possible 2^{64} output values. This provides the key to the attack.

Many other implementation details have been skipped, and would give rise to similar attacks. For instance, most of the additions require multiple-precision operations with carries. High-level languages such as C do not have an "add with carry" operation, so carries need to be implemented with some form of branch. This may provide the required difference in execution path; such things may occur with probabilities ranging from a few times in 2^{64} to once in 2^{32}, all of which provide a suitable basis for an attack.

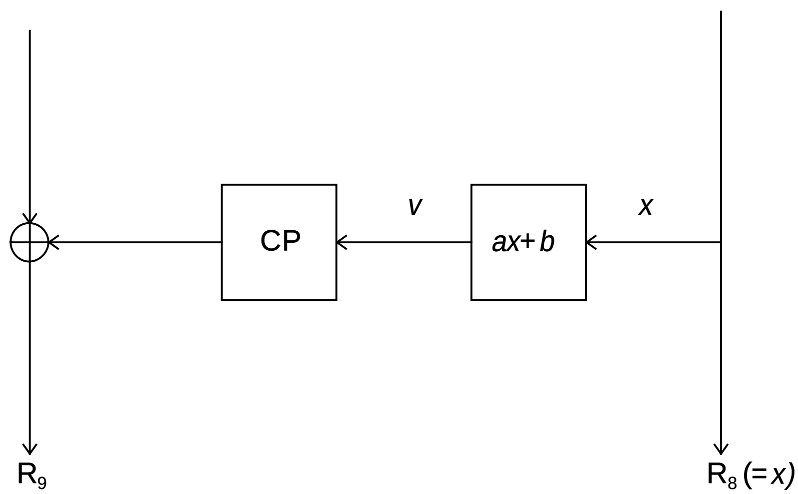
3. Mounting the attack

In order to mount an attack on DFC, we need first to select an event *E* associated with an execution path producing a limited range of ($ax+b$ mod (W+13)) values. Call the set of such values *V*. In addition, construct a set *D* of all possible differences between elements of *V*:

$D = \{\ d\colon d = v_i - v_j \bmod (W+13),\ v_i \in V,\ v_j \in V,\ d \neq 0\ \}.$

Next we observe the DFC implementation and collect the ciphertexts produced whenever E occurs in the final round.

Final Round



From the ciphertext, we obtain a set of x_i such that $v_i = a\, x_i + b \bmod (W+13)$ where $v_i \in V$.

Taking the first two such values, x_1 and x_2, we obtain the simultaneous equations

$$ax_1 + b = v_1 \quad \bmod (W+13)$$

and

$$ax_2 + b = v_2 \quad \bmod (W+13)$$

Subtracting,

$$a(x_1 - x_2) = v_1 - v_2 \bmod (W+13) = d,\ d \in D.$$

We now solve the equation $g(x_1 - x_2) = 1 \bmod (W+13)$, and the possible candidates for a are given by $a = gd \bmod (W+13)$, for all $d \in D$.

The procedure is repeated again with the ciphertexts x_2 and x_3 to give another g and thus candidate list of a values. The real value of a will appear in all candidate lists; after enough tries only one will remain.

This attack also provides a means of detecting success or failure (i.e. either one a will remain after several tries, or no such a will be found), so it is possible to eliminate incorrect x_i values if the procedure for observing them is not totally reliable.

Using this attack, there is no obvious way to directly recover b and thus strip off the final round. Note, however that the range of possible values for b is greatly reduced (for all x_i, $(ax_i + b) \in V$), which means that brute-force searching for b in conjunction with an attack on event E occurring in the previous round may become possible. If so, the entire cipher can be undone.

4. Analysis

This attack depends on observing event E a sufficient number of times. A randomly picked value of $(ax_i + b)$ will fall in set V with probability approximately $|V|/2^{64}$. Depending on the implementation, not all occurrences of this will cause event E. The probability of E in a given ciphertext is then $\alpha|V|/2^{64}$, where $\alpha \leq 1$.

For each event E (not counting the first) we will eliminate the majority of the keyspace for a, ending up with a fraction $|D|/2^{64}$ left. Assuming each E eliminates an independent fraction, it will take $1 + \log(2^{64})/\log(2^{64}/|D|)$ events before we have one candidate.

For most implementations, the values of $v \in V$ are in a simple arithmetic progression - in general c, $c+d$, $c+2d$, $c+3d$ ... $c+(m-1)d$, where c and d are constants, and $m = |V|$). For such values, the values in the difference set D will be $\pm d$, $\pm 2d$ .. $\pm(m-1)$d, and the value of |D| is $2|V|-1$.

Plugging these values in, if we can observe an event E based on 32-bit operations which occurs (say) whenever the high or low word of a 64-bit value is zero, we have $|V| = 2^{32}$ and $\alpha=1$. E occurs once on average every 2^{32} ciphertexts, and we need about 4 E events to find a unique candidate. The attack will therefore take approximately 2^{34} encryptions to recover a.

For a rare event E (e.g. one which occurs 13 times out of 2^{64} with a likelihood α of 0.5), we have a probability of $6.5 \,.\, 2^{-64}$ per ciphertext, and we need (rounded up) 3 events to find a. This requires of the order of 2^{63} plaintexts - still less expensive than brute force search.

5. Practicalities

Many practical attacks have been published against cryptography in smartcards based on measuring such things as overall execution time, or current consumption as a function of time. It is realistic to suggest that event E would be observable in a typical smartcard implementation. It is also likely that a dedicated hardware implementation could contain an observable E if steps were not taken to eliminate it.

Observing the event *E* occurring in the algorithm executing on a typical desktop machine may not be practical. The small variations in execution time or current consumption will most often be masked by other activity (cache fills, interrupts, context switches, etc.). However, note that event *E* implies a different path through the code is being taken, and this occurs with very low probability. The following “bug attack” scenario becomes a possibility:

Suppose the code implementing event *E* has a bug in it, which is not discovered during testing as it occurs so rarely. When it does occur, some time later, some failure will result - an incorrect ciphertext block will be sent causing a connection to be dropped, or the machine may crash - which may be observable by an attacker. Alternatively an attacker may be able to tamper with an implementation such that *E* is easily observable in some way, but such a modification will not be detected when standard test vectors are applied.

6. Conclusion

While not weakening the theoretical basis of the DFC cipher, this paper has illustrated that a practical implementation may contain weaknesses which lead to an attack. These weaknesses stem from a key part of the cipher algorithm, and mean that extreme care must be taken when an implementation is designed. In particular, in a software implementation this may require the use of assembly code rather than a high-level language.

Of equal concern is the fact that not all possible execution paths through the core of the cipher code (or, not all gates in a hardware implementation) will necessarily be executed during testing. Firstly, this requires extreme care during design to ensure it is correct, and secondly, it may be impossible to produce a set of standard test vectors which validate all parts of the design, and therefore tampering with the design may go undetected.

7. References

1 S. Vaudenay, ‘Decorrelated Fast Cipher: an AES Candidate’, `ftp://ftp.ens.fr/pub/dmi/users/vaudenay/GG+98b.ps`