Sender: Guillaume.Poupard@ens.fr
Date: Fri, 09 Apr 1999 10:19:24 +0200
From: Guillaume Poupard <Guillaume.Poupard@ens.fr>
Organization: Ecole Normale Superieure
X-Mailer: Mozilla 3.04 (X11; I; SunOS 5.7 sun4u)
To: aesfirstround@nist.gov
CC: Serge Vaudenay <Serge.Vaudenay@ens.fr>
Subject: Implementation of DFC on low cost smart cards

Please find attached to this e-mail the final Postscript version of our paper "Decorrelated Fast Cipher: an AES Candidate well suited for low cost smart cards applications" of CARDIS'98, co-authored with Serge Vaudenay. This "OFFICIAL Comment" may be of some interest for the evaluation of the implementability of DFC on various systems.

Regards,

Guillaume Poupard

--
Guillaume POUPARD (DMI/GRECC) || mailto:Guillaume.Poupard@ens.fr
Ecole Normale Superieure || http://www.dmi.ens.fr/~poupard/
45, rue d'Ulm || +33-01 44 32 20 48
75005 Paris, France || +33-01 44 32 20 80 (fax)

Decorrelated Fast Cipher: an AES Candidate well suited for low cost smart cards applications

Guillaume Poupard and Serge Vaudenay

Ecole Normale Supérieure, Laboratoire d'informatique
45 rue d'Ulm, F-75230 Paris Cedex 05, France
email: {Guillaume.Poupard,Serge.Vaudenay}@ens.fr

Abstract. In response to the call for candidates issued by the National Institute for Standards and Technologies (the Advanced Encryption Standard project) the Ecole Normale Supérieure proposed a candidate called DFC as for "Decorrelated Fast Cipher", based on the decorrelation technique that provides provable security against several classes of attacks (in particular the basic version of Biham and Shamir's Differential Cryptanalysis as well as Matsui's Linear Cryptanalysis). From a practical point of view, this algorithm is naturally very efficient when it is implemented on 64-bit processors. In this paper, we describe the implementation we made of DFC on a very low cost smart card based on the Motorola 6805 processor. The performances we obtain prove that DFC is also well suited for low cost devices applications.

Since the beginning of commercial use of symmetric encryption (with block ciphers) in the seventies, construction design used to be heuristic-based and security was empiric: a given block cipher was considered to be secure until some researcher published an attack on.

The Data Encryption Standard [1] initiated an important open research area, and some important cryptanalysis methods emerged, namely Biham and Shamir's differential cryptanalysis [4] and Matsui's linear cryptanalysis [11], as well as further generalizations. Nyberg and Knudsen [14] showed how to build toy block ciphers which provably resist differential cryptanalysis (and linear cryptanalysis as well as has been shown afterward [3]). This paradigm has successfully been used by Matsui in the MISTY cipher [12, 13]. However Nyberg and Knudsen's method does not provide much freedom for the design, and actually, this paradigm leads to algebraic constructions. This may open the way to other kind of weaknesses as shown by Jakobsen and Knudsen [8].

In response to the call for candidates for the Advanced Encryption Standard (AES) which has been issued by the National Institute of Standards and Technology (NIST) the ENS proposed in [6] the *Decorrelated Fast Cipher* (DFC)[1]. It is a block cipher which is faster than DES and hopefully more secure than triple-DES. It accepts 128-bit message blocks and any key size from 0 to 256. We believe that it can be adapted to any other cryptographic primitive such as

[1] See http://www.dmi.ens.fr/~vaudenay/dfc.html

stream cipher, hash function, MAC algorithm. The new design of DFC combines heuristic construction with provable security against a wide class of attacks. Unlike the Nyberg-Knudsen paradigm, our approach is combinatorial. It relies on Vaudenay's paradigm [15–19]. This construction provides much more freedom since it can be combined with heuristic designs.

In [6] we provided proofs of security against some classes of general simple attacks which includes differential and linear cryptanalysis. This result is based on the decorrelation theory. We believe that this cipher is also "naturally" secure against more complicated attacks since our design introduced no special algebraic property. Our design is guaranteed to be vulnerable against neither differential nor linear cryptanalysis with complexity less than 2^{81} encryptions. We believe that the best attack is still exhaustive search. Another theoretical result claims that if we admit that no key will be used more than 2^{43} times, then the cipher is guaranteed to resist to any iterated known plaintext attack of order 1.

From a practical point of view, the main computations are an affine mapping $x \mapsto P = ax + b$ where a, b and x are 64-bit operands and P a 128-bit quantity, followed by two reductions modulo $2^{64} + 13$, and modulo 2^{64} respectively. Modern computers, like those of the AXP family, have properties that make the implementation of DFC especially efficient because of there native 64-bit processor. As an example, we are able to encrypt 500 Mbps using the new Alpha processor 21264 600MHz provided that the microprocessor can input the plaintext stream and output the ciphertext stream at this rate (see [7]).

The aim of this paper is to describe the implementation we made of DFC on a very low cost smart card based on a 8-bit processor Motorola 6805, using less than 100 bytes of RAM and without the help of any kind of crypto-processor. This proves that DFC is well suited for a large range of applications.

Section 1 gives a high level overview of DFC (a full description can be found in [6]). Section 2 explains how to efficiently deal with the multiprecision arithmetic needed to implement the algorithm. Finally, section 3 exposes our implementation and the performances we obtained.

1 Definition of DFC

Notations

All objects are bit strings or integers. To any bit string $s = b_1 \dots b_\ell$ we associate an integer $\overline{s} = 2^{\ell-1}b_1 + \dots + 2b_{\ell-1} + b_\ell$ The converse operation of representing an integer x as an ℓ-bit string is denoted $|x|_\ell$. The concatenation of two strings s and s' is denoted $s|s'$. We can truncate a bit string $s = b_1 \dots b_\ell$ (of length at least n) to its n leftmost bits $\mathrm{trunc}_n(s) = b_1 \dots b_n$.

High Level Overview

The encryption function DFC operates on 128-bit message blocks by means of a secret key K of arbitrary length, up to 256 bits. Encryption of arbitrary-

length messages is performed through standard modes of operation which are independent of the DFC design (see [2]).

The secret key K is first turned into a 1024-bit "Expanded Key" EK through an "Expanding Function" EF, *i.e.* $\mathrm{EK} = \mathrm{EF}(K)$. The EF function performs a 4-round Feistel scheme (see Feistel [5]).

The encryption itself performs a similar 8-round Feistel scheme. Each round uses the "Round Function" RF. This function maps a 64-bit string onto a 64-bit string by using one 128-bit string parameter.

Given a 128-bit plaintext block PT, we split it into two 64-bit halves R_0 and R_1 so that

$$\mathrm{PT} = R_0 | R_1$$

Given the 1024-bit expanded key EK, we split it into eight 128-bit strings

$$\mathrm{EK} = \mathrm{RK}_1 | \mathrm{RK}_2 | \ldots | \mathrm{RK}_8$$

where RK_i is the ith "Round Key".

We build a sequence $R_0, \ldots, R_9$ by the Equation

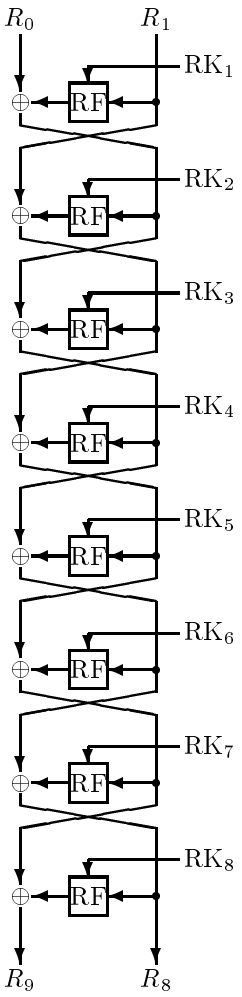
$$R_{i+1} = \mathrm{RF}_{\mathrm{RK}_i}(R_i) \oplus R_{i-1} \quad (i = 1, \ldots, 8)$$

We then set $\mathrm{CT} = \mathrm{DFC}_K(\mathrm{PT}) = R_9 | R_8$.

The RF function (as for "Round Function") is fed with two 64-bit parameters, a and b. It processes a 64-bit input x and outputs a 64-bit string. We define $\mathrm{RF}_{a|b}(x) =$

$$\mathrm{CP}\left(\left|\left(\left(\bar{a} \times \bar{x} + \bar{b}\right) \bmod (2^{64} + 13)\right) \bmod 2^{64}\right|_{64}\right)$$

where CP (as for "Confusion Permutation") is a permutation over the set of all 64-bit strings. It uses a look-up table RT (as for "Round Table") which takes a 6-bit integer as input and provides a 32-bit string output.



Let $y = y_l | y_r$ be the input of CP where y_l and y_r are two 32-bit strings. We define

$$\mathrm{CP}(y) = \left|\overline{(y_r \oplus \mathrm{RT}(\overline{\mathrm{trunc}_6(y_l)}))} | (y_l \oplus \mathrm{KC}) + \overline{\mathrm{KD}} \bmod 2^{64}\right|_{64}$$

where KC is a 32-bit constant string, and KD is a 64-bit constant string.

In order to generate a sequence $\mathrm{RK}_1, \ldots, \mathrm{RK}_8$ from a given key K represented as a bit string of length at most 256, we first pad K with a constant pattern KS in order to make a 256-bit "Padded Key" string by $\mathrm{PK} = \mathrm{trunc}_{256}(K | \mathrm{KS})$. This allows any key size from 0 to 256 bits.

Then, the key scheduling algorithm consists in using the previously defined encryption algorithm reduced to 4 rounds instead of 8. This enables an efficient and secure (see [6]) generation of expanded keys without increasing the size of the program. We can observe that the key scheduling algorithm complexity is the same as the complexity of the encryption of four 128-bit blocks of data.

On the fly encryption and decryption

The expanded key may be too large (128 bytes) to be stored in the RAM of some low cost smart cards. Anyway, it can be computed on the fly during each block encryption in such a way that only two round keys RK_i are stored simultaneously in memory. This makes the algorithm 5 times slower because the key setup requires 8 encryption using a 4-round version of DFC.

This strategy can also be used for decryption using a minor modification of the key scheduling algorithm based on the precomputation of the last two round keys RK_7 and RK_8.

2 Multiprecision Arithmetic

The internal operations deal with 64- and 128-bit numbers so we have to implement multiprecision arithmetic. The key operations, used to compute RF, are an affine mapping $x \mapsto P = ax + b$ where a, b and x are 64-bit operands and P a 128-bit quantity, followed by two reductions modulo $2^{64} + 13$, and modulo 2^{64} respectively.

The implementation of the multiprecision multiplication follows a classical scheme, without any optimization, such as Karatsuba's, which would not be worthwhile for so small operands. For the modular reduction, we use the following method. First we write

$$P = Q \times 2^{64} + R \tag{1}$$

where R is the remainder of the Euclidean division of P by 2^{64} (no computation is required for this). Then we rewrite (1) as:

$$P = Q \times (2^{64} + 13) + R - 13 \times Q \tag{2}$$

As we want to reduce modulo $2^{64} + 13$, the quantity $Q(2^{64} + 13)$ disappears. However, this is not yet perfect as we have to deal with two cases: $R - 13Q > 0$ and $R - 13Q < 0$. To avoid this, we rewrite (2) as:

$$P = (2^{64} + 13) \times (Q - 13) + (13 \times (2^{64} - 1 - Q) + 182 + R) \tag{3}$$

In (3), the quantity $2^{64} - 1 - Q$ is just the bitwise complement of Q. The modular reduction thus consists of the easy evaluation of:

$$P' = 13 \times (2^{64} - 1 - Q) + 182 + R \tag{4}$$

modulo $2^{64}+13$. The result is always positive and, most of the time, greater than $2^{64}+13$. We thus perform a second reduction, using the same formula. We rewrite (4)

$$P' = Q' \times 2^{64} + R' \tag{5}$$

and we define P'' as

$$P'' = 13 \times (2^{64} - 1 - Q') + 182 + R' \tag{6}$$

The result P'' verifies $0 \leq P'' < 2 \times 2^{64} + 13$ so the final value of the modular reduction is P'' if $P'' < 2^{64}+13$ and $P''-(2^{64}+13)$ otherwise. These comparison and subtraction can be performed very efficiently because of the very special form of the binary representation of $2^{64}+13$.

Timing Attacks

Implementations must be such that computations of the modular multiplication does not leak any information by time measurement (*i.e.* we must avoid tests and conditional branches that depend on the computation). Otherwise this may leak some information by Kocher's timing attacks [9].

The previous description of the modular reduction shows that the only step that does not require a fixed number of clock cycles is the last one. Consequently we always compute $P'' - (2^{64}+13)$ even if the result is note always used.

The main problem with timing attacks appears during the computation of $P = ax + b$. Using the classical scheme for multiplication, differences of timing may appear because of carry propagation. A simple but not really efficient solution consists in removing all the conditional branches by propagating the carries as far as possible even if they are zeros.

3 Implementation on Smart Cards

We have implemented DFC on a cheap smart card based on a 8-bit processor Motorola 6805 running at 3.57 MHz. This microprocessor can perform very simple operations with two 8-bit registers. For example it can only perform rotations of one bit. Furthermore, a byte multiplication is implemented.

The smart card possesses three kinds of memory, a ROM of 4 KB, an EEPROM of 2 KB where the program and the data are stored and a 160-bytes RAM where only 120-bytes can be used. Furthermore, the stack is automatically set in its middle and consequently the available memory is not contiguous. The communication rate is 19200 bits per second and the server is a simple PC.

Two implementations of DFC are proposed:

- the first one performs the key scheduling just once and stores the expanded key EK (1024 bits) in RAM. It needs 171 bytes of memory.
- the second one needs 89 bytes of RAM but is 5 times slower because it computes the expanded key during encryption of each block. Anyway, it may be interesting for some applications to reduce the size of the RAM and accordingly the cost of the used smart cards.

The amount of ROM needed to store the program and constant data is less than two kilo-bytes. More precisely, the program needs 1600 bytes, half of which are used by the multiplication $P = ax + b$ for efficiency reasons. Consequently, the size of the program could be easily reduced at the cost of a slower execution.

The DFC algorithm depends on several constants whose total size is 348 bytes. In order to convince that this design hides no trap-door, we choose the constants from the hexadecimal expansion of the mathematical e constant. Furthermore, the size of the data can be reduced to 268 bytes since a part of the RT table can be reused for other constants.

The time needed to encrypt a 128-bit block is about 40.000 cycles (resp. 200.000 cycles for the second version). With a clock frequency of 3.57 MHz, our implementation is able to encrypt 1428 bytes per second (resp. 285 bytes per second). This is twice slower than the best known implementation of triple-DES on the same platform and such a rate is similar to the amount of data the card can exchange with the server per second.

	Version **1**	Version **2**
RAM used	171 bytes	**89** bytes
Size of the program (see the appendix)	1600 bytes	
Size of data	268 bytes	
Size of the needed EEPROM (or ROM)	$<$ **2** KB	
Cycles to encrypt a 128 bit block	**40 000** cycles	200 000 cycles
Key setup (precomputation of EK)	160 000 cycles	~ 0
Blocks encrypted per second at 3.57 MHz	89 blocks	18 blocks
Data encrypted per second at 3.57 MHz	1428 bytes	285 bytes
Time to encrypt a 128 bit block at 3.57 MHz	11 ms	56 ms

Performances of two implementations of DFC on a 6805 based smart card

4 Conclusion

Even if DFC seems to be designed for powerful computers, the implementation we made of this algorithm on a very low cost smart card proves that it is well suited for a large range of applications. Hardware-based implementations require to implement the multiplication which may be painful for the designer, but efficiently possible. Although we did not investigate all possible applications, we believe that there is no restriction on the implementability of DFC.

References

1. Data Encryption Standard. *Federal Information Processing Standard Publication 46*, U. S. National Bureau of Standards, 1977.
2. DES Modes of Operation. *Federal Information Processing Standard Publication 81*, U. S. National Bureau of Standards, 1980.
3. K. Aoki, K. Ohta. Strict evaluation of the maximum average of differential probability and the maximum average of linear probability. *IEICE Transactions on Fundamentals*, vol. E80-A, pp. 1–8, 1997.
4. E. Biham, A. Shamir. *Differential Cryptanalysis of the Data Encryption Standard*, Springer-Verlag, 1993.
5. H. Feistel. Cryptography and computer privacy. *Scientific American*, vol. 228, pp. 15–23, 1973.
6. H. Gilbert, M. Girault, P. Hoogvorst, F. Noilhan, T. Pornin, G. Poupard, J. Stern, S. Vaudenay. Decorrelated Fast Cipher: an AES Candidate. Submitted to the call for candidate for the Advanced Encryption Standard issued by the National Institute of Standards and Technology.
7. O. Baudron, H. Gilbert, L. Granboulan, H. Handschuh, R. Harley, A. Joux, P. Nguyen, F. Noilhan, D. Pointcheval, T. Pornin, G. Poupard, J. Stern, S. Vaudenay. DFC Update. In *the Proceedings from the Second Advanced Encryption Standard Candidate Conference*, National Institute of Standards and Technology, 1999.
8. T. Jakobsen, L. R. Knudsen. The interpolation attack on block ciphers. In *Fast Software Encryption*, Haifa, Israel, Lectures Notes in Computer Science 1267, pp. 28–40, Springer-Verlag, 1997.
9. P. Kocher. Timing attacks in implementations of Diffie-Hellman, RSA, DSS and other systems. In *Advances in Cryptology CRYPTO'96*, Lectures Notes in Computer Science 1109, pp. 104–113, Springer-Verlag, 1996.
10. L. R. Knudsen, B. Preneel. Fast and secure hashing based on codes. In *Advances in Cryptology CRYPTO'97*, Santa Barbara, Californie, U.S.A., Lectures Notes in Computer Science 1294, pp. 485–498, Springer-Verlag, 1997.
11. M. Matsui. The first experimental cryptanalysis of the Data Encryption Standard. In *Advances in Cryptology CRYPTO'94*, Santa Barbara, Californie, U.S.A., Lectures Notes in Computer Science 839, pp. 1–11, Springer-Verlag, 1994.
12. M. Matsui. New structure of block ciphers with provable security against differential and linear cryptanalysis. In *Fast Software Encryption*, Cambridge, Royaume Uni, Lectures Notes in Computer Science 1039, pp. 205–218, Springer-Verlag, 1996.
13. M. Matsui. New block encryption algorithm MISTY. In *Fast Software Encryption*, Lectures Notes in Computer Science 1267, pp. 54–68, Springer-Verlag, 1997.
14. K. Nyberg, L. R. Knudsen. Provable security against a differential cryptanalysis. *Journal of Cryptology*, vol. 8, pp. 27–37, Springer-Verlag, 1995.
15. S. Vaudenay. Provable security for block ciphers by decorrelation. In *STACS 98*, Lectures Notes in Computer Science 1373, pp. 249–275, Springer-Verlag, 1998.
16. S. Vaudenay. The decorrelation technique home-page. URL:`http://www.dmi.ens.fr/~vaudenay/decorrelation.html`
17. S. Vaudenay. Adaptive-Attack Norm for Decorrelation and Super-Pseudorandomness. Tech. report LIENS-99-2, Ecole Normale Supérieure,1999.
18. S. Vaudenay. On the Lai-Massey Scheme. Tech. report LIENS-99-3, Ecole Normale Supérieure, 1999.
19. S. Vaudenay. Resistance against General Iterated Attacks. To appear in *Advances in Cryptology EUROCRYPT' 99*, Prague, Czech Republic, Lectures Notes in Computer Science 1592, Springer-Verlag, 1999.