A Fault Attack Against the FOX Cipher Family

L. Breveglieri[1], I. Koren[2], and P. Maistri[1]

[1] Department of Electronics and Information Technology, Politecnico di Milano, Milano, Italy
{brevegli, maistri}@elet.polimi.it
[2] Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA, USA
koren@ecs.umass.edu

Abstract. Since its first introduction, differential fault analysis has proved to be one of the most effective techniques to break a cipher implementation. In this paper, we apply a fault attack to a generic implementation of the recently introduced FOX family of symmetric block ciphers (also known as Idea Nxt). We show the steps needed to mount an effective attack against FOX-64. Although the basic characteristics of this cipher are similar to those of AES, FOX uses a non-invertible key schedule which makes it necessary to use a different attack plan. We also estimate the number of faulty ciphertexts required to reveal the secret key. Our results can be easily extended to other variations of the cipher that use longer inputs and keys.

1 Introduction

Most recent cryptosystems are now designed to be secure against common attack techniques, such as linear or differential cryptanalysis. To this end, encryption algorithms are often made public, allowing the research community to analyze them and find possible weaknesses. As a result, the attackers' attention has been shifted to the actual implementations of cryptosystems, which can leak useful information about the secret key (e.g., simple and differential power analysis).

Recently, a technique exploiting errors injected during the encryption (or decryption) process proved to be a very effective attack. In [4], the authors showed how a single faulty encryption is enough to break a CRT-RSA cryptosystem; in [2], faults injected into a DES architecture were successfully used to recover the secret key. Since then, fault-based attacks have been applied to a variety of cryptosystems: public-key based ones (ECC, XTR), stream ciphers (RC4) and block ciphers.

Initially, there was skepticism about the feasibility of these fault-based attacks, until in [10] the authors showed that even with very cheap equipment (a microscope and a camera flash) they were able to change the stored values in static RAM cells. Nowadays, smart cards are tested by manufacturers to study their vulnerabilities to fault attacks using specialized laser equipment. Obviously, laser beams increase the chance of a successful fault attack compared to

L. Breveglieri et al. (Eds.): FDTC 2006, LNCS 4236, pp. 98–105, 2006.

a simple camera flash, since the attacker can control more precisely parameters like wavelength, energy, duration and location.

Most research efforts in the area of fault-based attacks have focused on AES, due to its adoption as an NIST standard. In the first proposed attacks, the attacker was assumed to be able to flip single bits within the chip with very precise timing [3]. Such an assumption is getting harder to justify due to technological restrictions. As die size shrinks, the precision required to affect a single flip-flop requires expensive equipment (e.g., laser). Moreover, designers are beginning to consider fault attacks as a serious security risk and are introducing countermeasures that can overcome single bit errors.

Later, new types of attacks were proposed, were the fault model was relaxed and random byte errors were considered. In this scenario, the attacker is able to alter the value of a whole byte, possibly being able to either decide or know its location, but without having knowledge of its previous value. On the other hand, timing is still important, since imprecise injections are useless. One of the most impressive results is the attack published in [9], where only two precisely injected faults can break the AES-128 cipher.

Several countermeasures have already been proposed, mostly based on some form of redundancy. In [7], it is suggested to compute the inverse operation at the encryption, round or operation level. In [1,6,8] an error detecting code is used to protect the internal data path. In [11], a pipeline architecture is used to detect any transient fault. In addition to these techniques, the chip may also be protected by means of sensors or shielding.

So far, to the best of our knowledge, there is no published fault attack against the Idea Nxt cipher which has some unique characteristics. In this paper, we describe how to mount a successful fault-based attack against this cipher. In particular, we show the number of required faults that have to be injected, on average, to mount a successful attack and recover the key.

The paper is organized as follows. Section 2 describes the Fox family of ciphers, focusing on the details needed to understand the basics of the attack, which is described, step-by-step, in Section 3. Section 4 concludes the paper, summarizing the results and suggesting possible countermeasures.

2 The FOX Cipher Family

FOX is a family of symmetric block ciphers recently presented as Idea Nxt [5] and aimed at multimedia streaming and secure data storage. The cipher can be customized in terms of the block length (see Table 1), key size and number of iterations.

The algorithm contains repetition of the round function *lmor*, followed by a single instance of the function *lmid*. The latter differs in the absence of an additional automorphism applied to the leftmost part of the input block, which constitutes a single Feistel iteration (see Figure 1). Moreover, these two functions have a fixed high-level structure, but are customized in width and in the internal parameters to suit both Nxt64 and Nxt128.

Table 1. The FOX cipher family

Name	Input size	Key size	Iterations
NXT64	64 bits	128 bits	16
NXT128	128 bits	256 bits	16

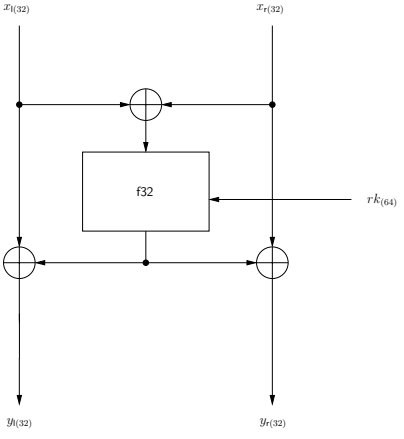

Fig. 1. *lmid* function

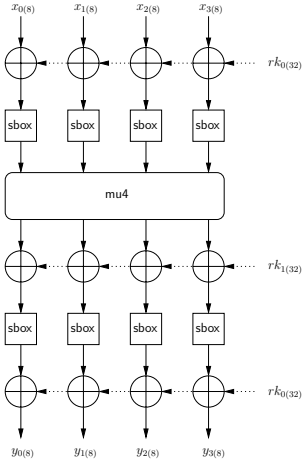

Fig. 2. The *f32* function used in Idea Nxt 64

The core part of every round iteration is the *f32* (*f64* in Idea Nxt128) function, which includes linear and non-linear operations (see Figure 2). In this function, the key is split into two parts: the left one is added at the beginning and at the end, while the right one is added in the central body, right after the diffusion step. There are two non-linear steps implemented by means of substitution boxes (Sboxes): the Sbox can be implemented either as a byte lookup table, or as a combination of three different smaller tables operating on 4-bit nibbles. This is similar to the AES Sboxes, which can be defined by using composite fields. However, unlike AES, an algebraic definition does not seem to exist here. The diffusion step is defined as a linear transformation over the Galois Field GF(2^8). The irreducible polynomial is $x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + 1$ which is different from the one used by AES.

The key schedule routine uses the same operations employed in the encryption datapath. The master secret key is updated in every iteration by using some precomputed constant. Each updated key is then used to generate a single round key for the encryption datapath, through a sequence of non-linear and linear operations. In addition, a compression stage exists, where bit pairs are exclusive ORed. This process is shown in Figure 3: the *sigma* block shown in the figure is the non-linear stage made of substitution boxes.

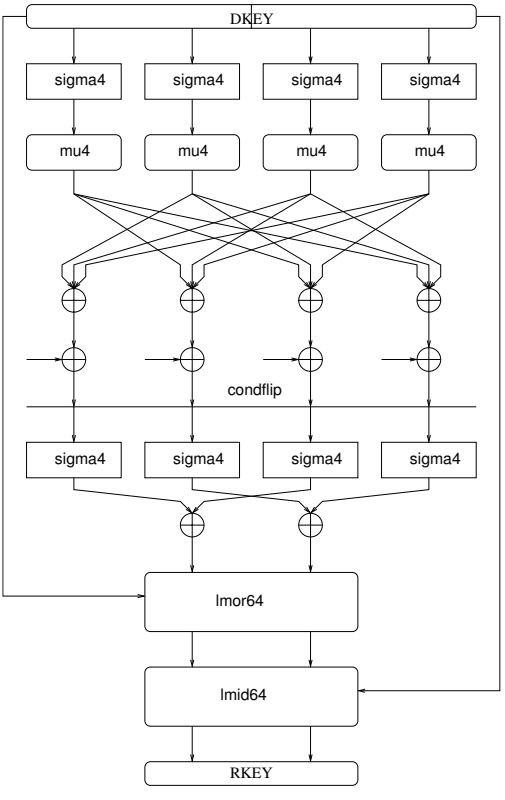

Fig. 3. Round key generation from secret key

3 The Attack

A Differential Fault Analysis (DFA) attack allows to apply differential and linear analysis to only a few rounds of the cipher, possibly only one. This allows to mount a very effective attack at an almost negligible cost. A conventional differential attack would consider the whole cipher (in terms of the number of iterations and size of the data block), which often leads to the ability to only attack reduced versions of the cryptosystems.

By collecting a few differential pairs relative to the last non-linear step, the attacker can reduce and finally guess the values computed in the last rounds, thus being able to infer the last round key. Once the key has been recovered, the key schedule can be inverted to obtain the initial secret key; if this is not possible, then the attack can be reiterated on each round, starting from the last, until the whole key material is exposed.

The key schedule for the FOX ciphers is non-invertible: hence, once the last round key is revealed, the attack will be repeated either on the preceding round or on the key schedule directly to recover the master secret key. In this extended

abstract we describe the initial steps required to mount the attack and recover the last round key. All the steps to recover the whole key material and the related results will be described in the full paper.

In the following, we will consider a fault as a random byte addition in GF(2^8), i.e., a XOR with a random byte value, such as in [9]. A single bit fault, in fact, would lead to a simpler problem, but is less realistic. A byte fault, on the other hand, can be injected more easily. Exact knowledge of the location of the affected byte is not crucial, although it simplifies the analysis. If this information is not available, then a guess can be made and verified later with additional experiments (i.e., faults) until a unique guess is possible. Timing, on the other hand, is very important: the fault injection must be carefully synchronized with the encryption process, in order to affect the desired operations. This can be achieved, for instance, by analyzing the power trace of the device while computing and identifying the desired round. The exact location within the round can be determined after few (random) attempts.

The attack on the last round must be planned in two phases, since the round key is actually used in two separate instances. The first phase allows to retrieve the leftmost part of the key, which is used at the beginning and at the end of the round. The fault must be injected before the last non-linear operation, which means between the first and the last Sbox stages of the final round. Any time instance in this interval is fine, but the effectiveness of the attack is increased if the injection occurs before the linear diffusion step, i.e., the *mu4* operation. In this case, the linear transformation spreads the fault through the whole word and more information is provided.

For instance, suppose that an error ϵ is injected into the leftmost byte of the word, right before the *mu4* operation, resulting in the error word $(\epsilon, 0, 0, 0)$. Then, the error is spread by the diffusion step and the error word becomes $(\epsilon, \epsilon, c\epsilon, \alpha\epsilon)$, where c and α are coefficients of *mu4*. This is the differential input to the last substitution operation, and although unknown, we can still identify some regularity, and prune for instance all those values which are not admissible for each byte of the word (i.e., error values that would give an empty set of candidates for the Sbox inputs). The output differential, on the other hand, is known and this information can be used to narrow the search.

The first fault injection is used to build the set of all possible candidates, considering any admissible fault value. For each additional fault, a new candidate set is built and intersected with the current solution set, thus narrowing the number of possible candidates. The process continues until a unique candidate is identified for each possible byte. The value found in this way is the input to the last Sbox step. Thus, it is easy to recover the key value used in the last key addition, i.e., the left part of the last round key.

In this phase, knowing where the fault has been injected simplifies the analysis, because we know how the error spreads after the *mu4* operation. If this is not the case, however, the actual location may be guessed and the analysis performed as described above; for each additional fault, a new guess is made. If the guesses are all correct, then the procedure will give the unique desired solution. If an

empty candidate set is found, then at least one of the guesses was incorrect and we have to backtrack and try another possible solution. Although this procedure increases the complexity of the search tree, we found that the correct key value could be often identified after only 2 or 3 attempts when the location was known. The whole search tree, with a new branch for each location guess and for each injected fault, gives 4^f leaves where f is the number of faults. This number is an upper bound, since many branches can be pruned after each fault injection, thus reducing the complexity of the attack.

The second phase aims at recovering the rightmost part of the round key. The approach is the same and is based on injecting a fault before the first non-linear step of the last round. In this case, however, the structure of the last round (see Figure 1) gives us both the input and the output differentials, making the analysis much easier. On the other hand, the diffusion step does not provide any additional information, which means that each byte must be targeted individually.

Based on our simulations, the last round key was completely revealed after 11.45 injected faults on average. Further analysis of the distribution of the faults required to recover the key reveals that the first phase requires from 2 to 8 faults, while the second phase uses from 8 to 28 faults: the worst case is however rare, and the average values are about 2.94 and 8.51 for the first and second phase, respectively. The complete attack requires from 8 to 31 faults. The distribution curves are shown in Figure 4 where the worst cases (when more than 20 faults are required) are not shown for clarity, but they constitute a negligible percentage of the overall test space (less than 0.02%).

If the fault location is unknown, we are confident that a few fault injections may be still enough to identify the leftmost part of the key. This issue does not arise when performing the second phase of the attack on the round, since the injected error can be inferred from the output result (see Figure 1). This phase, however, requires more faults since we cannot exploit the diffusion properties of the linear stage. In fact, each byte of the key must be attacked separately. In the full paper, we will provide the number of faults required to reveal the whole key material in the two main scenarios.

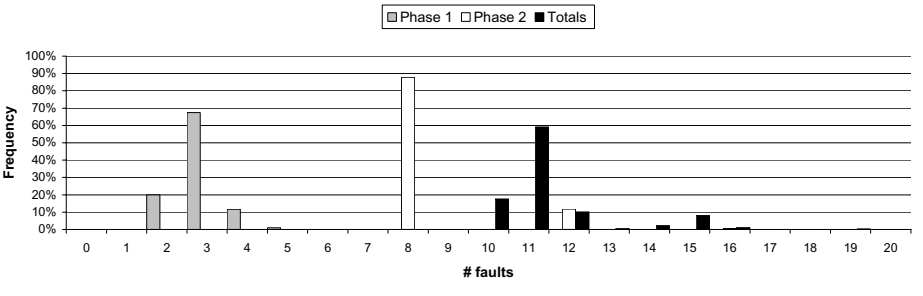

Fig. 4. Number of fault injections required to mount the attack and recover the key: left half (phase 1), right half (phase 2) and whole key (totals)

4 Conclusions

In this paper, we present a fault injection attack against the newly introduced FOX family of ciphers. The attack resembles currently known attacks against AES, but unlike AES the key schedule of Idea Nxt is not invertible. This forces the attacker to iterate the fault injection on every round of the encryption algorithm to recover the whole key material, or to attack the key schedule directly, as will be shown in the full paper.

The last round key can be found, on average, after 11.45 faulty encryptions. If we assume that attacking any single round or the key schedule has the same complexity, then the whole cipher can be broken after 183 or 23 faults, respectively. These results will be confirmed in the full paper.

Differential fault analysis proves to be one of the most effective attack techniques and can be used when the attacker has the ciphering device even for a short time. Generic countermeasures such as those presented in [7,11] or shielding and sensors are possible. Moreover, the cipher is based on GF arithmetic and can therefore be protected by means of a parity code, such as in [1]. We plan to implement the cipher in hardware and evaluate the effectiveness and overhead of these countermeasures.

References

1. G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and V. Piuri. "Error analysis and detection procedures for a hardware implementation of the advanced encryption standard," *IEEE Trans. Computers*, 52(4):492–505, 2003.
2. E. Biham, A. Shamir, "Differential Fault Analysis of Secret Key Cryptosystems," Technical Report, Technion - Computer Science Department, 1997.
3. J. Blömer, J.-P. Seifert, "Fault Based Cryptanalysis of the Advanced Encryption Standard (AES)," Financial Cryptography, *Lecture Notes in Computer Science*, vol. 2742, pp. 162-181, 2003.
4. D. Boneh, R. DeMillo, R. Lipton, "On the Importance of Eliminating Errors in Cryptographic Computations," *Journal of Cryptology*, vol. 14, pp. 101-119, 2001.
5. P Junod and S. Vaudenay. "FOX : A New Family of Block Ciphers," Selected Areas in Cryptography, 11th International Workshop, SAC 2004, Lecture Notes in Computer Science, vol. 3357, pp. 114-129, Springer, 2004.
6. M. G. Karpovsky, K. J. Kulikowski, and A. Taubin, "Robust protection against fault-injection attacks on smart cards implementing the advanced encryption standard," *2004 International Conference on Dependable Systems and Networks (DSN 2004), Proceedings*, pages 93–101. IEEE Computer Society, 2004.
7. R. Karri, K. Wu, P. Mishraand and Y. Kim. "Concurrent error detection schemes for fault-based side-channel cryptanalysis of symmetric block ciphers," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 21(12):1509–1517, Dec 2002.
8. R. Karri, G. Kuznetsov, and M. Gössel. "Parity-based concurrent error detection in symmetric block ciphers," *Proceedings 2003 International Test Conference (ITC 2003)*, pages 919–926. IEEE Computer Society, 2003.

9. G. Piret, J.-J. Quisquater, "A Differential Fault Attack Technique against SPN Structures, with Application to the AES and Khazad," Cryptographic Hardware and Embedded Systems - CHES 2003, *Lecture Notes in Computer Science*, vol. 2779, Springer-Verlag, pp. 77-88, 2003.
10. S. P. Skorobogatov and R. J. Anderson, "Optical Fault Induction Attacks," Cryptographic Hardware and Embedded Systems - CHES 2002, Lecture Notes in Computer Science, vol. 2523, pp. 2-12, Springer, 2003.
11. K. Wu and R. Karri. "Idle cycles based concurrent error detection of RC6 encryption," *16th IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT 2001), Proceedings*, pages 200–205. IEEE Computer Society, 2001.