UCL Crypto Group Technical Report Series

A timing attack against Rijndael

François Koeune and Jean-Jacques Quisquater

http://www.dice.ucl.ac.be/crypto/

Technical Report
CG–1999/1

Place du Levant 3
B-1348 Louvain-la-Neuve, Belgium

Phone: (+32) 10 472541
Fax: (+32) 10 472598

A timing attack against Rijndael

François Koeune and Jean-Jacques Quisquater

June 10, 1999

Département d'Électricité (DICE), Université catholique de Louvain
Place du Levant 3, B-1348 Louvain-la-Neuve, Belgium
E-mail: {fkoeune, jjq}dice.ucl.ac.be

1 Introduction

Besides classical cryptanalysis, searching weaknesses in a cryptosystem by observing it like a mathematical object, recent research has also focused on attacks aimed at physical implementations of the cryptosystem. These types of attacks, among which we can for example cite fault attack [BDL97, Len96], timing attack [Koc96, DKL+98], simple and differential power analysis [KJJ98, MDS99, BS99, CJRR99], ... turned out to be more restrictive – as they work only against specific implementations of a cryptosystem –, but also much more efficient than their "classical" correspondings, in the sense that the amount of resources needed to carry them out is often much smaller.

This paper describes a timing attack against the AES candidate Rijndael. We show how a careless implementation can be broken with some thousands measures per key byte, provided a very limited knowledge of the implementation.

2 Brief description of Rijndael

A complete description of Rijndael can be fond in [DR98]. We will focus here on the parts of interest for the attack.

A Rijndael encryption consists of an initial round key addition, followed by N round transformations, the last round being slightly different from the others. The different transformations applied during each round operate on an array of bytes, called the *state*, composed of 4 lines and N_b columns (where N_b is the block size, in 32-bit words).

Basically, each round, except the last one, consists of the following steps:

CG–1999/1

1. *ByteSub*, which will operate a fixed substitution on a byte-by-byte basis; each byte will thus be transformed independently of the value of other bytes of the state;

2. *ShiftRow*, which will permute some bytes of the state without modifying their value;

3. *MixColumn*, which will be described later;

4. *AddRoundKey*, which will add a round key to every byte of the state.

The *MixColumn* transformation operates on the columns of the state and applies them the following matrix multiplication:

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

where multiplications is defined in $GF(2^8)$ as multiplication of binary polynomials modulo the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$.

3 The vulnerable model

Due to the choice of the matrix and irreducible polynomial, *MixColumn* can be implemented very efficiently: first, it is easy to see that, as '03'='02'+'01', the only multiplications that will actually have to be performed are by '02'; second, it can be showed ([DR98]) that multiplication by '02' in GF(256) can be implemented very fast as follows:

- shift byte one position left,
- if a carry occurs, XOR the result with '1B'.

If the implementation is careless, the above operation will thus not be constant time, but will take longer in the case were a carry occurs. Throughout the rest of this paper, we assume such a behaviour.

4 Basic idea

Now, let us focus on what will happen to a given byte (say, the first) of a known plain text during the first few encryption sub-steps:

- before entering the first round, that byte will be XOR-ed with a byte of round key – call it R_1 –, whose value is unknown, but constant independently of the plain text;
- then a substitution S, according to a known S-box, will take place;
- the byte will then be moved around (to a known place) by *ShiftRow*, without being modified;
- finally, *MixColumn* will occur; during this operation, the byte will be multiplied at least once by '02'; we will denote by ξ this specific occurrence of the multiplication; as we have seen, it will be longer if the first bit of the byte is set.

If we were able to observe the time taken by that peculiar multiplication, we would thus immediately be able to deduce the first bit of the round key. Of course, this is not the case, as we have no access to partial operations, but only to the total encryption time. However, if we encrypt a large amount of random messages, we can expect all the other operations to behave as random noise and therefore hope to be able to "filter out" the information we are interested in.

5 The attack

5.1 Initialization phase

We begin by building a table describing, for every possible first key byte and for N different possible values of the first plain text byte*, if the multiplication ξ will require an additional XOR or not.

This table T with thus have $256 \times N$ entries as follows

$$\begin{aligned} \forall 0 \leq i \leq 255, 0 \leq j < N,\ T[i][j] &= 1 \text{ if the first bit of } ByteSub(i \text{ XOR } j) = 1, \\ &= 0 \text{ otherwise.} \end{aligned}$$

Each line i corresponds to a possible value for R_1, the first byte of the round key; each column j corresponds to a different value of the first plain text byte.

*We could of course build 256 different such values; experiments, however, show that the attack can be carried out with much less (typically, 20) values.

5.2 Measure phase

Similarly, we build N sets of M messages, where the first byte of every message from set S_i is equal to i; the other bytes are random. Thus, for every message from subset S_i, the multiplication ξ will be exactly the same.

Let us now encrypt these messages and measure the computation times. If M is large enough, we can expect the mean time for subset S_i to reflect the time taken by the multiplication ξ, i.e. to be slightly bigger when that multiplication is long. We have thus built an oracle that, with some error probability, determines, for i in $[0, N-1]$, if the first bit of $ByteSub(i \text{ XOR } R_1)$ is set.

To determine R_1, it suffices to compare the oracle with the array T: the line that best reflects the oracle's predictions should correspond to the right value of R_1. Other bytes of the first round key can be recovered in the same way by turning our attention to the second, third, ... bytes of plain text.

Due to Rijndael's key schedule, once N_k (the key size, in 32-bit words) consecutive bytes of round key are known, the complete round keys can be generated. We have thus broken the cipher in the case were the block size is greater or equal than the key size. If the block size is smaller, then a little more work is necessary, but the idea is just the same.

6 Practical results

We did not carry out a mathematical estimation of the success probability of the above oracle. However, we tried to use the above method against the 128-bit block, 128-bit key version of Rijndael.

Experiments showed that, with 3000 samples per key byte, the complete key was recovered with very high probability at negligible cost. It also appeared that there was still some information leaking even if much less samples were available; better techniques could therefore drastically improve these results.

7 Conclusion

This paper is nothing more than a new witness that implementation-aimed attack represent a very important threat, which must absolutely be taken into account in the design of a cryptographic application.

We insist on the fact that the result presented here is not an attack against Rijndael, but against bad implementations of it. We would also like to point out that the 8051 implementation performed by Rijndael's authors was implemented with timing attack in mind and turned out to be immune against the threat described here.

Therefore, we do not think this result to constitute an argument against- or pro- the adoption of Rijndael as the future AES. We simply claim that the implementations of this future standard – whichever it will be – will have to be performed with great care.

References

[BDL97] D. Boneh, R.A. DeMillo, and R.J. Lipton. On the importance of checking cryptographic protocols for faults. In W. Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, Konstanz, Germany*, volume 1233 of *LNCS*, pages 37–51. Springer, 1997.

[BS99] E. Biham and A. Shamir. Power analysis of the key scheduling of the aes candidates. In *Proc. second AES conference*, 1999.

[CJRR99] S. Chari, C. Jutla, J. Rao, and P. Rohatgi. A cautionary note regarding evaluation of aes candidates on smart cards. In *Proc. second AES conference*, 1999.

[DKL+98] J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater, and J.-L. Willems. A practical implementation of the timing attack. In J.-J. Quisquater and B. Schneier, editors, *Proc. CARDIS 1998, Smart Card Research and Advanced Applications*, LNCS. Springer, 1998.

[DR98] Joan Daemen and Vincent Rijmen. AES proposal: Rijndael. In *Proc. first AES conference*, August 1998. Available on-line from the official AES page: `http://csrc.nist.gov/encryption/aes/aes_home.htm`.

[KJJ98] P. Kocher, J. Jaffe, and B. Jun. Introduction to differential power analysis and related attacks. `http://www.cryptography.com/dpa/`, 1998.

[Koc96] P. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *Advances in Cryptology - CRYPTO '96, Santa Barbara, California*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996.

[Len96] A.K. Lenstra. Memo on RSA signature in the presence of faults. Available from the author, Sept. 28 1996.

[MDS99] T. S. Messerges, E. A. Dabbish, and R. H. Sloan. Investigations of power analysis attacks on smartcards. In *Proc. USENIX Workshop on Smartcard Technology*, 1999.