Resistance Against Implementation Attacks
A Comparative Study of the AES Proposals

Joan Daemen
Proton World Int'l
Zweefliegtuigstraat 10
B–1130 Brussel, Belgium
`daemen.j@protonworld.com`

Vincent Rijmen*
K.U.Leuven, Dept. ESAT
Kard. Mercierlaan 94
B–3001 Heverlee, Belgium
`rijmen@esat.kuleuven.ac.be`

February 1, 1999

Abstract

In this paper we consider timing attacks and power analysis attacks on smart card implementations of the AES candidate algorithms. We try to assess how difficult it will be to make the implementations of the algorithms resistant against these attacks.

1 Introduction

When ciphers are employed in real-world applications, not only mathematical attacks have to be considered. The ciphers are implemented in hardware or software, and attackers may choose to exploit weaknesses of the implementation rather than weaknesses of the cipher. In this paper we consider two recently described powerful attacks; the timing attack [1] and (differential) power analysis [2]. Both attacks work under the assumption that the attacker can monitor the working of the cryptographic device that executes the algorithm. We concentrate on implementations on 8-bit processors, but the comments also apply for implementations on other processors. Even a design in dedicated hardware can leak information and has to be reviewed carefully in order to be secure against these types of attacks.

In Section 2 we give a general description of the attacks. In Section 3 we describe what operations are necessary for the implementation of each of the 15 AES candidates (for HPC, we only consider the 'medium' variant, which is the variant that deals with block size 128) and what opportunities the operations might give to mount an attack. We conclude with detailed comments on the AES algorithms.

2 Discussion of the Attacks

We describe in short the attacks of [1, 2] and how they can be countered.

*F.W.O. Postdoctoral Researcher, sponsored by the Fund for Scientific Research - Flanders (Belgium)

2.1 Timing attacks

A timing attack can be mounted if the execution time of the cipher depends on the value of the key. Let us illustrate this by an example. Assume we have a cipher implementation in which an instruction is executed on the condition that a certain key-dependent value b has a specific value. If no special precautions are taken, the total execution time of the cipher will vary depending on whether the conditional instruction is executed or not. Hence, it is possible to deduct the value of b from careful observations, e.g., by comparing the cipher execution times for different values of b, while taking care that all other execution time influencing parameters are kept constant, or by averaging out the influence of other parameters.

An implementation can be protected against timing attacks by ensuring that the cipher execution time is independent of the value of the key. For conditional instructions e.g., this can be done by inserting NOP instructions in the shortest path until all paths take the same time. However, this solution might leave the cipher unprotected against power analysis attacks (cf. Section 2.2). A very obvious requirement in this context is that the number of cycles that an instruction takes to execute, should be independent of the value of the operands.

2.2 Simple power analysis

Let $P(I, op_1, op_2, \ldots, op_n)$ denote the power consumption pattern that is consumed by the chip if it executes instruction I, with input bits given by $op_1, \ldots op_n$. The consumed power depends on the particular instruction I and on the value of the inputs.

In general, the power consumption pattern of a given instruction with given input bits is not constant but typically depends on environmental parameters (supply voltage, temperature, ...) and processes that go on inside the semiconductor ("noise").

Unless special precautions are taken, the following instructions will have very different power consumption patterns:

- addition;
- bitwise boolean operation;
- storing the value of a register in RAM;
- storing the value of a register in EEPROM;
- loading a value from RAM/EEPROM to a register;
- ...

If the attacker can distinguish between instructions using this information, and if the sequence of instructions depends on the key, he may exploit this to gain information on the key.

The first requirement for an implementation to be secure against simple power analysis is that the sequence of instructions should be independent of the cipher key. (Except for instructions with indistinguishible power consumption patterns.) This implies that conditional branches are to be avoided, because it is usually difficult to hide which path of instructions is taken, and thus the leaking of the evaluation of the condition, which depends on intermediate values, cannot be prevented. The ability to code a cipher in a fixed sequence of clock cycles depends on the operations used in the definition of the cipher and the instruction set available. Furthermore it is important that the power used for the execution of an instruction, is independent of the value of the operands. But a discussion of this aspect brings us to the next section.

2.3 Differential power analysis

Differential power analysis is an implementation attack that exploits the fact that for some instruction I the average power consumption pattern is correlated to the value of at least one input bit (or a function of input bits). In order to simplify the discussion we assume in the following that the average power consumption P_I of an instruction I is correlated to bit op_1, but the analysis applies to any bit op_i. Let

$$P_I(0) = E_{op_2,\ldots,op_n}[P(I,0,op_2,\ldots,op_n)]$$

and

$$P_I(1) = E_{op_2,\ldots,op_n}[P(I,1,op_2,\ldots,op_n)].$$

Here $E_{op_2,\ldots,op_n}$ denotes averaging over all values of $op_2,\ldots,op_n$.

The fact that there is a correlation between P and op_1 implies that $P_I(0) \neq P_I(1)$. Let

$$D_I = P_I(0) - P_I(1). \tag{1}$$

The correlation between P_I and op_1 is more exploitable if $D(i)$ has a large amplitude in relation to the 'noise' that is caused by fluctuating conditions on the chip and/or special measures taken by the chip manufacturer.

A straightforward differential power attack could occur according to the following steps:

1. **Target specification phase:** Identify a bit a in the intermediate encryption result that will very probably appear as an input bit of an instruction, that is expected to have a large D_I for this bit. It is required that this bit can be computed from known bits, i.e., input, output or already known intermediate encryption result bits D, and unknown round key bits K, by applying a function. We have

$$a = f(K,D)$$

2. **Data collection phase:** collect power consumption patterns for a sample of n executions of the block cipher, taking care that the bits of D vary over the sample.

3. **Data analysis phase:** guess a value for K and partition the power consumption patterns in two groups: those for which $a=0$ and those for which $a=1$, for the given K. Accumulate the power consumption patterns for the two subsets and compute the difference between the results. Or mathematically:

$$T_I(K) = \sum_{f(K,D)=0} P(D) - \sum_{f(K,D)=1} P(D). \tag{2}$$

If the value for K is correct, $T_I(K) \approx D_I \cdot n/2 \neq 0$. If the value of K is wrong, the two sums at the right side of (2) contain both power consumption patterns corresponding with $a=1$ and $a=0$. Unless the partitioning corresponds with a particular difference in consumption pattern, the value of $T_I(K)$ is expected to be much closer to 0.

Thus, the key value K that maximizes $|T_I(K)|$ is likely to be the correct value. In practice, the function $f(K,D)$ may have a particular structure, causing $|T_I(K)|$ to peak at more than one value. Usually it is still possible to determine the correct key value by performing some extra analysis ewploiting the structure of f.

Observe that the attacker doesn't need to know how the input dependency is caused, nor does he need to know which instructions are exactly executed. He just has to collect data and look out for peaks in his $|T_I(K)|$ diagram. Any correlation between the selected bit and the power consumption will pop up all by itself.

Moreover, the attacker has control over the applied voltage, clock frequency and pulse shape, temperature and other environmental parameters. He can select these to his advantage.

2.3.1 Countermeasures

The countermeasures against differential power analysis are not hard to guess. There are basically two ways to approach it, and the best is to combine the two.

The first way is to prevent, or at least to seriously complicate the exploitation of correlations. This can be done by hardware or software design. Some examples:

Desynchronisation: The DPA attack described above is based on the silent assumption that the sequence of instructions is fixed, so that in the adding of the power consumption patterns, the instructions pair nicely. If the sequence is not hard-coded but changes from cipher execution to cipher execution, e.g., by inserting dummy instructions based on some modifying parameter, this straightforward attack no longer works. However, as every instruction has its own characteristic power consumption pattern, it may still be possible to get around the desync. Moreover, if this sequence scrambling is based on a secret key, this secret key may be very easy to retrieve using simple power analysis.

Software balancing: The program is written in such a way that words treated contain for each of the data bits its complement. In this way the correlation to single data bits is diminished.

Power consumption randomisation: A hardware module is built into the chip that adds noise to the power consumption. This reduces the signal-to-noise ratio of the attack and as such makes it harder.

The second way is to minimise the correlation between the data and the power consumption. If the cipher is implemented in software, this can be done by modifying the ALU hardware so that all instructions that operate on cipher key or input, are *balanced.* If one can afford a dedicated hardware implementation, this can be built in such a way that it is balanced. Basically, the design goal is similar in both cases.

Very probably, complete absence of correlation between input bits and power consumption pattern cannot be obtained. However, the technical feasibility of a power analysis attack strongly increases with the amplitude of the correlation. Moreover, the other measures are likely to be more effective when the correlation to be concealed is smaller.

In practice, all instructions operating on data or key bits used in a program implementing a block cipher should be balanced. Hence, in this context there is definitely an advantage in using a small set of instructions and not using instructions that are hard to balance.

3 The AES Candidates: Operations Used

Table 1 gives on overview of the used operations for each cipher. The exor operation, which is used by all the 15 candidates, is left out of the table.

	CAST-256	Crypton	DEAL	DFC
table-lookup	8 to 32	8 to 8/32	6 to 4	6 to 32
shift/rotate	32 var	32 (byte)*	multiple	-
logical ops.	and fix	and fix	and fix	not, and
addition	32	-	-	64
subtraction	32	-	-	-
multiplication	-	-	-	-
other	-	-	-	mult. $\bmod 2^{64}+13$
				add. $\bmod 2^{64}+13$

	E2	Frog	HPC	Loki97
table-lookup	8 to 8	8 to 8	-	11/13 to 8
shift/rotate	64 (byte)	-	64	64
logical ops.	or fix	-	or, and fix	and, or, not
addition	-	8	64	64
subtraction	-	-	64	64
multiplication	32	-	64	-
other	div 32	-	-	-

	Magenta	Mars	RC6	Rijndael
table-lookup	8 to 8	8/9 to 32	-	8 to 8/32
shift/rotate	-	32 var	32 var	32 (byte)
logical ops.	-	and, or	and fix	-
addition	-	32	32	-
subtraction	-	32	32	-
multiplication	-	32	-	-
other	-	-	squaring $\bmod 2^{32}$	-

	SAFER+	Serpent	Twofish
table-lookup	8 to 8	-	8 to 8/32
shift/rotate	8	32	32
logical ops.	-	and, or, not, ...	-
addition	8	-	32
subtraction	8	-	-
multiplication	-	-	-

Table 1: Overview of the operations in the 15 AES candidates. The shifts/rotates are over a fixed numner of positions, unless ‘var’ is added. The multiplication with 2 in SAFER+ and RC6 is assumed to be implemented as a (fixed) left shift. For the logical operations, ‘fix’ indicates that one of the operands of the operation is a constant.

3.1 Storing registers to memory

This instruction is not susceptible to a timing attack. With respect to current analysis, the problem is that the writing of a 0 and the writing of a 1 have a different current consumption. To counter this, one can hardwire the circuit so that if a word is written, also its complement is written. This implies that the word and its complement are available prior to being written.

3.2 Loading registers from memory

In loading registers from memory, one can obtain a better balancing by reading a word and its complement. This implies that the word and its complement must be available.

3.3 Table lookups

This instruction is not susceptible to a timing attack and very probably not to simple power analysis. A table-lookup operation is typically implemented with an indexed load instruction. In this operation there are two aspects:

- concealment of data in the load index register instruction;
- concealment of the address in a load accumulator instruction.

To compute the address, the table input must be added to the table off-set. In general, this is an arithmetic addition. However, this can be converted to simple concatenation on the condition that the table starts at an address that is a multiple of 2^n where n is the size of the input. In concatenation there is no interaction between the bits, so that the instruction can be balanced by simply hardwiring the addressing operation double: one for the address and the other for its complement.

3.4 Rotations and shifts

Shifts and rotations can be implemented using shift and rotate instructions or table-lookups. Using table-lookups allows to speed up the operation if shifts/rotations over more than a single bit is required and only a single-bit rotate/shift instruction is available. If a table-lookup mechanism is used in the implementation, obviously all requirements treated in the section on table-lookup apply.

3.4.1 Fixed offset

In rotations and shift operations with fixed offset, timing is independent of the key and therefore no threat. If the word length used in the cipher is equal to the processor word length, shift and rotate instructions can be used in a straightforward way. If not, they can be implemented using shifts (or rotations) over a single bit and using the carry mechanism. Although it presents no vulnerability against timing attacks, it is rather time-consuming.

To counter current analysis, it is a requirement that the used rotation and shift instructions are balanced. The bit interaction is slightly more complicated than in the case of load and store instructions. However, it can be expected that doubling the circuit to also include the complement of the register already gives some improvement. If 32-bit shifts and rotations are implemented on an 8-bit processor with single-bit shifts, it is only that single instruction (or the two, left and right shift) that needs to be balanced.

Some ciphers use fixed byte rotations. If implemented on an 8-bit processor, these rotations can be implemented without actually executing a 'rotate' or 'shift' instruction. This also implies that a rotation/shift can be decoupled in a bit shift part and a byte shift part. For instance a rotate over 11 bits can be implemented as a byte rotate followed by a rotate over 3 bits. It can be seen that no bit shifts over more than 4 bits are needed.

3.4.2 Data-dependent offset

If the number of bits that words are shifted or rotated over depend on the data, implementing the rotation or shift by means of single-bit shifts is not straightforward. To have a fixed sequence of instructions is certainly problematic. One can imagine that a left shift and a right shift can be made hard to distinguish, so that one can ensure that the number of shifts is always the maximum number and that in most cases there will be compensating left and right shifts (even in the case the shift offsets happens to be 0). In this way, shifts with an odd offset and those with an even offset will still be distinguishable from each other. If use is made of the byte shifts, an additional constraint is that the address computation shall be made balanced.

If implemented by means of table-lookup, a table is required for every possible shift amount (or for every shift amount modulo 8). Care has to be taken that the computation that is required to determine which table is selected does not leak information.

3.5 Bitwise boolean operations

In general, bitwise boolean operations have constant number of cycles hence there is no vulnerability with respect to timing attacks.

In a bitwise boolean operation, the bits of a word are combined two by two and there is no interaction between bits across the word. If the interactions between the different bits is neglected, The expression for the power consumption can be reduced to:

$$P_I(a_1, a_2, \ldots a_n, b_1, b_2, \ldots b_n) = \sum_i P_i(a_i, b_i)$$

hence it reduces to the sum of n independent functions, each in two variables. Balancing P_I is reduced to balancing the component functions $P_i(a_i, b_i)$ correlation.

This is valid in the case that the circuit for the bitwise boolean function needs to be implemented in an ALU, but also in a dedicated hardware implementation of the cipher, e.g., in the form of a cryptographic co-processor.

In practice a better balancing can be obtained by hardwiring the circuit for $aopb$ four times: $aopb$, $a'opb$, $aopb'$ and $a'opb'$. Obviously, one of the requirements is that the inputs (a, b) AND their complements (a', b') are available. Let $P_c(a, b)$ be the power consumption pattern corresponding to the circuit that implements $aopb$. We have:

$$P_i(a, b) = P_c(a, b) + P_c(a, b') + P_c(a', b) + P_c(a', b')$$

It follows that

$$P_i(0, 0) = P_i(0, 1) = P_i(1, 0) = P_i(1, 1)$$

Due to differences between the 4 circuits, the power consumption may still depend on the values. It can be seen that the values of the complements of all input bits must be available.

If this is also the case for the next instruction, the symmetry requirement implies that all four circuits must also provide the complement of the output bits.

Remark that logical operations with one fixed operand can easily be implemented with table lookups.

3.6 Arithmetic operations

In arithmetic operations there is a much more complex interaction between the bits of the two operands. The reasoning given for bitwise boolean operations is not applicable. In addition and subtraction, typically a carry propagation is implemented.

Balancing addition and subtraction by modifying the ALU hardware seems problematic. Balancing multiplication, with its intricate dependencies, seems even harder to achieve.

There are five ciphers among the AES contenders that do not use arithmetic operations on key or data values and therefore are not confronted with this problem: these are Crypton, DEAL, Magenta, Rijndael and Serpent.

4 Algorithm Specifics

The 15 algorithms can be devided into three groups.

Favorable: Algorithms that use only logical operations, table-lookups and fixed shifts, and that are therefore relatively easy to secure. The algorithms of this group are Crypton, DEAL, Magenta, Rijndael and Serpent.

Doubtful: Algorithms that use logical operations, table-lookups, fixed shifts and additions. The addition is suspected to be more difficult to secure. The algorithms of this group are Frog, Loki97, SAFER+, Twofish.

Problematic: Algorithms that use multiplications or variable rotations, operations that are expected to be difficult to secure. The algorithms of this group are CAST-256, DFC, E2, HPC, Mars, RC6.

4.1 CAST-256

CAST-256 uses exor, table-lookups, additions, subtractions, variable rotations, and 'and' with a fixed mask value. Both the arithmetic operations and the variable rotations may prove difficult to protect against implementation attacks. CAST-256 is a 'problematic' algorithm.

4.2 Crypton

Crypton uses a very small instruction set: exors, table-lookups, byte shifts and 'and' with a fixed mask value. Crypton sits in the group of the 'favorable' algorithms. The key scheduling of the latest version, Crypton v1.0, uses bit shifts over 2 and 6 bits. These can also be protected against implementation attacks, at the cost of some additional tables.

4.3 DEAL

DEAL uses the same small instruction set as the DES, which ranks it among the 'easy' algorithms. Furthermore, an implementation of DEAL can benefit from built-up expertise with DES implementations. This is not a big surprise, since reusing expertise is the whole purpose of the cipher.

4.4 DFC

DFC uses exors, table-lookups, 'and', modular additions and multiplications. The implementation of the multiplication and addition $\bmod 2^{64} + 13$ uses several multiplications, additions, table-lookups, bitwise inversion and subtractions. The multiplication seems very difficult to protect against differential power analysis, placing the cipher in the 'problematic' group.

4.5 E2

E2 uses exors, table-lookups, 'or' with a fixed mask value, multiplication and division. It is in the class of 'problematic' algorithms. The multiplication and the division are used only once: in the whitening of the input and the output. This makes it very important to protect these operations. An interesting question here is whether the security gained by performing this complicated whitening operations outweighs the extra cast to design an implementation attack resistant multiplication and division, if that is possible at all.

4.6 Frog

Frog uses mainly exors and table-lookups, which should be relatively easy to protect against implementation attacks. However, the key setup uses additions, placing Frog in the middle category.

4.7 HPC

HPC uses exors, additions, subtractions, multiplications, variable rotations and 'and' and 'or' with mask values. The wide range of instructions make the cipher expensive to protect. HPC is a problematic algorithm.

4.8 Loki97

Loki97 uses table-lookups, shifts, bitwise logical operations, additions and subtractions. It is in the middle category.

4.9 Magenta

Magenta uses only exors and table-lookups. It is probably the cipher that is the easiest to protect against implementation attacks among the 15 candidates. Alas, the cipher has other problems.

4.10 Mars

Mars uses exors, table-lookups, bitwise logical operations, additions, variable rotations, subtractions and multiplications. A strong feature of the cipher is the presence of a mixing phase at the start and the end of the cipher. The two mixing phazes effectively shield the cryptographic core. The mixing phases contain no variable rotations and are therefore probably less vulnerable to an implementation attack. On the other hand, the cipher starts with 4 additions and ends with 4 subtractions, which makes it absolutely necessary to protect these operations. Until the strength of the mixing phases is demonstrated, we think it is best to rank Mars among the 'problematic' algorithms.

4.11 RC6

RC6 uses exors, variable rotations, 'and' with a fixed mask, additions, subtractions and squaring modulo 2^{32}. The squaring operation and the variable rotations place RC6 firmly in the category of problematic algorithms.

4.12 Rijndael

Rijndael uses exors, table-lookups and the key schedule also uses fixed byte rotations. The cipher is in the group of 'favorable' algorithms.

4.13 SAFER+

SAFER+ uses exors, table-lookups, fixed rotations, additions and subtractions. The cipher is in the category of 'doubtful' algorithms.

4.14 Serpent

Serpent uses fixed shifts and rotations and bitwise logical operations (in bitslice mode). This makes it one of the most easy ciphers to protect.

4.15 Twofish

Twofish uses exors, table-lookups, rotates and additions. The presence of the additions place it in the 'doubtful' category.

5 Conclusions

We have tried to attract attention to the relevance of timing and current analysis attacks in the real-world and the differences in the AES candidates in their ability to be implemented in a resistant way. We have identified five algorithms that behave favorably in this respect. These are Crypton, DEAL, Magenta, Rijndael and Serpent.

In this paper we have not addressed the efficiency of the proposals. Still, a comparative study of the efficiency of the different algorithms on platforms that allow the described attacks should take into account the measures to be taken to thwart these attacks.

Obviously, our conclusions are only indicative. We advise to ask feedback from semiconductor manufacturers on the feasibility of circuit balancing.

References

[1] P. Kocher, “Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems,” *Advances in Cryptology, Proceedings Crypto'96, LNCS 1109*, N. Koblitz, Ed., Springer-Verlag, 1996, pp. 104–113.

[2] P. Kocher, J. Jaffe, B. Jun, “Introduction to Differential Power Analysis and Related Attacks,, available from `http://www.cryptography.com/dpa/technical/`.