Power Analysis of the Key Scheduling of the AES Candidates

Eli Biham[1] Adi Shamir[2]

[1] Computer Science Department
Technion – Israel Institute of Technology
Haifa 32000, Israel
Email: biham@cs.technion.ac.il
WWW: http://www.cs.technion.ac.il/~biham/

[2] Department of Applied Mathematics and Computer Science
The Weizmann Institute of Science
Rehovot 76100, Israel

Abstract. The winner of the AES competition is likely to be practically unbreakable by direct cryptanalytic techniques, and thus the choice should be influenced by secondary considerations such as speed, code size, and resistance to indirect attacks. In this paper we consider Kocher's power analysis technique, which targets smart card inplementations and extracts their key by analysing their power consumption graphs. We develop a new variant of this technique, which analyses the power consumed during the key scheduling part of the block cipher. What makes this part a particularly important target for power analysis is the fact that its power consumption is a function of the key but not of the data, and thus we can use it to attack fielded smart card systems in which the cleartexts, ciphertexts, protocols and implementation details are unknown to the attacker. The attack is extremely effective against block ciphers with DES-like key scheduling structure, but less effective against block ciphers with Skipjack-like key scheduling structure. After describing the new technique and proposing it as an additional evaluation criterion, we analyse the vulnerability of each one of the AES candidates to this attack.

1 The Basic Technique

In 1998, Paul Kocher published a technical note (available from his WWW home page www.cryptography.com) describing simple and differential power attacks on smart card implementations of various cryptographic schemes. The attacks were easy to carry out and surprisingly effective in practice.

The basic idea behind the attacks is the observation that the power consumed by the smart card at any particular time during the cryptographic operation is related to the instruction being executed and to the data being processed. For example, multiplication consumes more power than addition, and writing 1's consumes more power than writing 0's. At a higher level, even a simple visual inspection of the graph describing the power consumed by an RSA coprocessor can often reveal whether a number is squared or multiplied, and thus disclose the successive bits of the secret exponent. An even more powerful attack on iterated cryptosystems is based on the differential power analysis of a large number of encryptions with known cleartexts. By guessing some of the key bits entering the first round, the attacker can often divide the sampled population according to the value they (presumably) write into memory at the end of the first round. A correct guess of key bits will result in subpopulations which use markedly different amounts of electricity during and shortly after the time of writing, while an incorrect guess of key bits will result in indistinguishable subpopulations.

An important disadvantage of the approach described in Kocher's original note is that the attacker must know all the cleartexts (if he attacks the subkey of the first round) or all the ciphertexts (if he attacks the subkey of the last round). In reality, many smart card applications (used by banks, credit card companies, cellular telephone operators, pay-TV broadcasters, ID token manufacturers, etc) are known to use particular cryptosystems (such as DES), but the attacker does not typically know the precise value of the input (if it contains randomness, chaining values, or just unknown format) or of the output (in case it is protected by an error correcting code, or MAC'ed, or used in an unknown protocol).

In the new variant of power analysis proposed in this paper, the attacker does not have to know either the inputs or the outputs of the encryption operations, and does not even have to know the precise timing (relative to the beginning of the protocol) in which they were carried out, or the details of their software implementation in the smart card. For the sake of simplicity, we assume that the protocol always performs the same subprotocols in the same order, and that they always require the same number of clock cycles (these requirements can be greatly relaxed by using more sophisticated attacks). As a result, we can assume that the power consumption graphs of different executions of the protocol can be aligned and compared at the level of single instructions.

The first goal of the attacker is to discover the tiny sections of the power consumption graph which are related to the key scheduling parts of the encryption operations. He performs this task in two steps:

1. He executes the protocol a large number of times on a single smart card in different contexts. For example, he pays different amounts at different times, responds to different challenge values during computer logon, or initiates different cellular phone calls from different locations. He aligns and compares the multiple power consumption graphs at each clock cycle, and eliminates from further consideration those clock cycles in which the graphs show a large variability in the power consumption, due to differences in the processed data. The remaining clock cycles represent operations which are data independent.
2. He repeats step 1 for several smart cards, which presumably contain different cryptographic keys, and finds their common data independent regions. From these regions, he eliminates the clock cycles which have small variability among the different cards. The eliminated subregions are likely to be standard systemwide operations, which are unrelated to the particular key

used in a particular card.

The data collected by the attacker can be summarized by a three dimensional array. The three dimensions represent the execution number, the card ID, and the clock cycle, and the contents of each cell represents the precise current consumed by a particular execution of the protocol on a particular card at a particular time. The attacker performs for each clock cycle a two dimensional filtering process on the power consumption signals, looking for similarities in one dimension and variability in the other dimension. The filtering process identifies the relatively few clock cycles in which each card behaves in the same way (regardless of the values being processed) but different cards behave in different ways (depending on the card's unique identity). By definition, these periods will include the key scheduling elements of the encryption process, but not the routine bookkeeping operations or the actual data encryption steps.

The problem of identifying the key scheduling periods among the millions of clock cycles can also be reduced to manageable size by other means (e.g., by visually recognizing specific patterns of power consumption, such as the 16 nearly-identical rounds of DES encryption), but the proposed technique seems to be universal and easy to automate.

The next stage of the attack is to study the individual steps of the key scheduling algorithm. Software implementations of iterated block ciphers in 8 bit smart cards usually compute each subkey just before it is used in the appropriate round, since their small RAM's make it difficult to extract all the subkeys in advance. The subkeys are computed and stored in RAM in chunks of 8 bits, and the amount of current consumed during the write operation is related to the number of 1's among the 8 written bits. The measurements are likely to contain many errors and to provide only partial information on the written data, but they can be partially smoothed by averaging the power consumed by a large number of executions with a common key, since the power consumption at this stage is unaffected by the unknown data differences in the various encryptions. The crucial observation behind the new attack is that in many key scheduling structures, even imperfect measurements of the Hamming weights of a small number of bytes generated during the key scheduling part of the block cipher can completely reveal the key by using a very simple algorith which does not require key enumeration.

For the sake of concreteness, consider the key scheduling structure of DES. It has 16 rounds, and in each round 6 subkey bytes are computed. The total number of subkey bytes being written into the RAM is 96, and knowledge of the Hamming weight of each one of these bytes yields a linear equation (over the integers) in the 56 variables $x_1, \ldots, x_{56}$ representing the individual bits of the key. In principle, we have sufficiently many equations even if we ignore the integrality or the 0/1 nature of the solution. The only remaining questions are whether the equations are sufficiently independent to define a unique solution, and how to find it efficiently when the list of Hamming weights contains some errors.

Fortunately (or unfortunately), the structure of the key scheduling algorithm in DES is ideally suited for such analysis. The uneven cyclic shifts and the presence of the second permuted choice (PC2) imply that the grouping of original key bits into subkey bytes is random looking, and in fact the 96 equations we get have a full rank of 56, and are thus uniquely solvable for any right hand side values of Hamming weights.

To handle the practical problem of imperfect knowledge of the Hamming weights, we can proceed in several ways:

1. Use standard techniques from error correcting codes, exploiting the fact that we have 96 equations in only 56 variables.
2. Use the quirk in the design of the key scheduling algorithm in DES, that the key is broken into two halves, and each subkey byte contains only bits from one of these halves. The system of equations thus decomposes into 48 equations containing only the 28 variables representing one half of the key, and 48 equations containing only the 28 variables representing the other half of the key. Each subset of 28 binary values can be exhaustively enumerated, and the computed vectors of 48 Hamming weights which are sufficiently close to the observed vector of 48 Hamming weights (up to the allowed measurement errors) can be easily identified.

This analysis shows that DES (or triple-DES, or 100-fold DES) have design characteristics which make their key scheduling algorithms particularly vulnerable to this new type of power analysis. It is interesting to note that a more modern design such as Skipjack (which was designed specifically for smart card and PC card implementations), generates its subkeys in a way which makes it less vulnerable to this kind of attack: Each subkey consists of 4 bytes which are directly copied from the 10 bytes of the key in a fixed order and without any processing. Thus, the attack outlined in this paper yields 128 highly redundant equations, which provide just the Hamming weight of each one of the 10 key bytes. Each Hamming weight measurement yields about 2.54 bits of information, and thus even perfect measurement can reveal at most 25.4 bits of information on the 80-bit key. Due to likely errors in the Hamming weight measurements, the number of revealed bits is likely to be much smaller.

In the rest of this paper we perform a comparative study of the resistance of the 15 AES submissions to this type of attack.

2 Power Analysis of the Key Scheduling of the AES Candidates

2.1 CAST-256

The main operations in the key scheduling of Cast-256 (and also the main encryption operations) are the 8x32-bit S boxes. We assume that the S box operations are performed by four table lookups (each fetching a single byte), and that the attacker can receive the Hamming weight of each one of the bytes. In this case, the attacker gets full information on the input to the S box: He receives the

Hamming weight of the input (entropy 2.54 bits), and the Hamming weight of each of the output bytes ($4 \cdot 2.54$ bits). This suffices to identify the 8-bit input, even if there are small measurement errors. Even if he receives the Hamming weight of only four of the bytes (i.e., only of the output bytes) he can still find the input with high probability.

Consequently, Cast-256 is particularly vulnerable to the new attack: We can apply power analysis to the key schedule parts to find the key, and apply it to the encryption steps to find the actual inputs and outputs of all the encryption operations carried out in the smart card.

2.2 Crypton

The first operation in the key schedule of Crypton applies π to the key, where π mixes the words in the form of $A \wedge M$, $M \in \{3fcff3fc_x, fc3fcff3_x, f3fc3fcf_x, cff3fc3f_x\}$. When split into bytes, we can receive the Hamming weight of subsets of six bits of the key, from which we can derive the Hamming weights of consecutive pairs of key bits.

Thus, we get half of the bits of the key (where the Hamming weight of a pair is 0 or 2), and the parities of the rest of the pairs. In total we get information on 3/4 of the key, i.e., we miss only about 32 bits out of a 128-bit key. We can complete our information on the key by analyzing the output of the subsequent operations in the key schedule (σ, γ, and τ).

2.3 Deal

Deal uses DES as its basic operation. Thus, the analysis of DES described above is directly applicable, and finds all the subkeys in the same way.

2.4 DFC

The first steps in the key schedule of DFC XOR the key with four different constants, and keep the results in memory for a more complex processing. Therefore, each byte is written to memory four times, each time XORed with some other constant. The first write gives us 2.54 bits of information on the byte. Each of the following three writes adds some information, although it is smaller than 2.54 bits (as, for example, the same parity is given by all four writes). Therefore, we can find information equivalent to half or more of the key bits just by analysing this stage.

2.5 E2

The initial operations in the key schedule of E2 apply an S box to each of the bytes of the key, and then apply linear mixings (XOR) to the resultant bytes. Thus, we get 2.54 bits of information on each byte of the key from the output of the S boxes, and then we reduce the entropy given the Hamming weights of the results of the mixings. Thus we get about half of the key information from the first step of the key schedule.

2.6 Frog

Omitted.

2.7 HPC

Omitted.

2.8 LOKI97

LOKI97 applies expansion, permutations, and S boxes to the key in the first stage. Thus, we can get about 2.54 bits of information on each byte of the key from the first layer of S boxes, which is increased later by analyzing the second layer.

2.9 Magenta

Magenta's key schedule does not apply any operations to the key before encryption. The first operation involving the key mixes it with the upper half of the data, and thus the method described in this paper is not directly applicable.

2.10 Mars

The first step of the key schedule of Mars mixes the key bytes and constants, and thus we can gain about 2.54 bits of information on every byte. However, it is difficult to recover the actual key bits from this information since each written byte may depend on many key bytes in a complicated way. In the subsequent steps of the key schedule the least significant 9 bits of each word (of which we know the Hamming weight of 8 bits) are used as an index to an S box which is then mixed into the next word. Again, it is not straightforward to find the original key, due to the mixing of many key bytes. If there are small measurement errors of the Hamming weight, the attacker's information is even harder to use. It appears that in order to find the key, the attacker should guess in advance many of the key bits.

2.11 RC6

As in Mars, the key bytes are mixed with other key bytes in their first use, and thus it is more complicated to translate the knowledge on the Hamming weight of the results into key bit values.

2.12 Rijndael

Rijndael applies several layers of XORing each key word into the next word, with almost no other transformation in between. Thus, there is some mixing of the key bytes, but the derivation of the key from Hamming weight measurements is expected to be easier than in the cases of Mars and RC6.

2.13 Safer+

The key schedule of Safer+ computes the sums of each key byte with 16 different known constants. Therefore, it is expected that given the Hamming weights of all the 16 sums, the key bytes can be identified (almost) uniquely by simple analysis.

2.14 Serpent

The key schedule of Serpent applies a linear feedback shift register to the key, and then applies (bitsliced) S boxes to the result. Thus, it is expected that (as in the case of Mars and RC6), it will not be easy to derive useful information on the key from the Hamming weights of the results.

2.15 Twofish

The computation of the key schedule of Twofish is also complex and does not seem to reveal direct information on the key bits from Hamming weight measurements.