Side Channel Cube Attacks on Block Ciphers

Itai Dinur and Adi Shamir

Computer Science department
The Weizmann Institute
Rehobot 76100, Israel

Abstract. In this paper we formalize the notion of *leakage attacks* on iterated block ciphers, in which the attacker can find (via physical probing, power measurement, or any other type of side channel) one bit of information about the intermediate state of the encryption after each round. Since bits computed during the early rounds can be typically represented by low degree multivariate polynomials, cube attacks seem to be an ideal generic key recovery technique in these situations. However, the original cube attack requires extremely clean data, whereas the information provided by side channel attacks can be quite noisy. To address this problem, we develop a new variant of cube attack which can tolerate considerable levels of noise (affecting more than 11% of the leaked bits in practical scenarios). Finally, we demonstrate our approach by describing efficient leakage attacks on two of the best known block ciphers, AES (requiring about 2^{35} time for full key recovery) and SERPENT (requiring about 2^{18} time for full key recovery).

1 Introduction

State of the art block ciphers such as AES [1] and Serpent [2] were specifically designed to resist all the standard types of attacks such as differential and linear cryptanalysis (see [3] and [4]). These ciphers were extensively analyzed over many years and are widely believed to be theoretically secure (i.e. there is no cryptanalytic attack which can break them faster than exhaustive search). However, most of the practical implementations of these ciphers can be broken with feasible complexity by side channel attacks, which exploit partial information leaked during the encryption process. Typical examples of such side channels include counting the time taken by the encryption subroutine, physical probing of one of the wires in the processor, measuring the power consumption when data is written into memory, picking up electromagnetic radiation generated by the chip, etc. All these techniques violate the black box assumption about the encryption process, and make the cryptanalysis much easier.

Many papers on side channel attacks such as [5] concentrate on the *physical phenomenon* used to obtain this side information, and on *countermeasures* which minimize this particular type of leakage. In this paper we ignore these issues, and concentrate on generic techniques which can exploit any type of leaked information, regardless of how it was obtained (similar but more abstract approaches

are described in [6] and [7]). We consider the class of iterated block ciphers, in which the same physical hardware is used to sequentially execute all the rounds during the encryption process, and thus we assume that the same kind of side information is provided about the internal state after each one of the rounds. For example, if the attacker uses a fine needle to physically probe the lsb of the state register, it makes no sense to assume that he can get this information after the fifth round, but not after the first or ninth round.

We call the mapping from full internal states to leaked bits the *leakage function*. The basic model of *leakage attack* assumes that the attacker has a large number of (known or chosen) pairs of plaintexts and ciphertexts, and in addition he is given the value of the same leakage function at the end of each one of the rounds in each one of the encryptions. The simplest kind of leakage function is the i-th state bit (for some constant i), but obvious generalizations can provide more bits (e.g., a full byte, or the lsb's of all the state words) or a more complicated Boolean function of the internal state (e.g., the Hamming weight of one of the state words, or whether it was zero).

The data model in leakage attacks is a combination of the parallel kind of information available in block ciphers and the serial kind of information available in stream ciphers, and resembles the capital letter I (see Fig 1). In standard attacks on block ciphers, we usually assume that the attacker knows the initial plaintext and the final ciphertext, but nothing in between. In the case of stream ciphers, we usually assume that the attacker knows after each clock cycle one bit of information about the internal state which is provided by the output function, but does not know the initial state and there is no final state. Our data model provides the attacker with both types of information simultaneously. Note that the distinction between the various data models becomes fuzzy when we consider stream ciphers such as LEX [8] which are based on the round function of some block cipher, since in this case the leakage from the block cipher is similar to the sequence of outputs of the stream cipher.

Since more data is available to the attacker, leakage attacks are potentially easier than standard attacks. On the other hand, we cannot use many of our standard tools such as differential and linear cryptanalysis, since we cannot partially decrypt the (mostly unknown) state after a few rounds in order to exploit a high probability distinguisher.

Our basic approach is to exploit the plaintext and the information leaked in the first few rounds, and to ignore the ciphertext and the information leaked during later rounds of the encryption process. In particular, any state bit after the first round is a very simple function of a small number of plaintext and key bits. This is good in the sense that it is easy to analyze the function, but bad in the sense that it cannot provide any information about most of the key bits. As we consider more advanced rounds, we get increasingly complicated functions, but also the possibility of determining more key bits. As we shall see later, for many block ciphers we can find some early round in which the avalanche of key bits is complete, but the function is still a relatively low degree multivariate polynomial. This situation is an ideal starting point for the recently proposed

cube attack [9], which is an improvement and generalization of several previous algebraic attacks such as AIDA [10].

Before analyzing advanced types of leakage attacks, we must first analyze the performance of the simplest brute force key recovery attack. In the standard data model, brute force attacks recover the correct key by exhaustively searching the entire key space. However, when a single bit of information on an early encryption or decryption round is available to the attacker, faster attacks may exist due to an incomplete diffusion of the key into the state. For example, assume that the leakage function is just the value of the first state bit of AES with a 128-bit key. After one AES round, each one of the state bits depends on exactly 32 key bits of the initial key and 1 key bit of the first round key. If the attacker knows these leaked bits for 33 plaintexts, he can start by performing exhaustive search over all the possible values of the 33 key bits, and expects to find one partial key which is compatible with the data. He can then complete the attack in $O(2^{96})$ time by exhaustively searching only over the remaining bits of the initial key, which is faster than a full exhaustive search but still impractical. If the attacker was allowed to have four leaked bits from each state, he could repeat this process and find the other three chunks of 32 key bits in $O(2^{32})$ time, but this is not allowed in our basic data model which assumes that only one bit of information is available about each round in each encryption. To get a faster attack, the cryptanalyst should analyze the state bit after the second round. It is a complex function of all the 128 key bits, but its multivariate polynomial representation has a relatively low degree. This is exactly the situation that cube attacks can handle in a generic way, without having to tailor the attack to each particular scenario.

The rest of this paper is organized as follows. In Section 2 we provide a brief description of cube attacks. In Section 3 we explain the difficulty in applying the original cube attack when the leaked data is even slightly noisy, and develop a new type of robust cube attack which can overcome a large fraction of errors in the leaked data. In Section 4 we describe the results we obtained when applying the cube-based leakage attack to Serpent, and show that it is much faster than the best known non-cube attack in the same data model. In Section 5 we perform the same kind of analysis for AES, and in Section 6 we conclude and list some open problems.

2 Cube Attacks

In this section we give a brief description of the original cube attack. For a more detailed description, refer to [9].

Cube attacks are generic key derivation attacks which can be applied automatically to any cryptosystem in which even a single bit of information can be represented by a low degree multivariate polynomial in the key and plaintext variables. An interesting property of cube attacks is that they can be applied even when this polynomial is completely unknown (e.g., when the attacker probes a random wire in a dense chip, and does not know which signal it carries) or

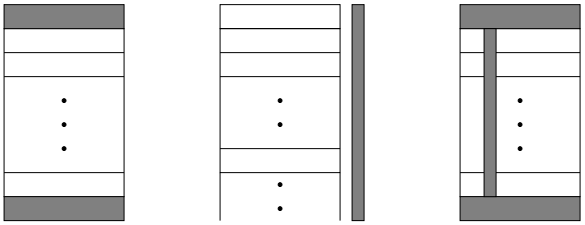

Fig. 1. The data model in leakage attacks (on the right) is a combination of the parallel kind of information available in block ciphers (on the left) and the serial kind of information available in stream ciphers (in the middle)

cannot be explicitly computed due to its length (and thus cannot be provided as input to other algebraic attacks such as XL [11] or Grobner bases [12]).

In almost any cryptographic scheme, each output bit can be described by a multivariate master polynomial $p(x_1, .., x_n, v_1, .., v_m)$ over $GF(2)$ of secret variables x_i (key bits), and public variables v_j (plaintext bits in block ciphers and MACs, IV bits in stream ciphers). The cryptanalyst is allowed to tweak the master polynomial by assigning chosen values for the public variables, which result in derived polynomials, and his goal is to solve the resultant system of polynomial equations in terms of their common secret variables. The cube attack is an algorithm for solving such polynomials, as described next.

To simplify our notation, we now ignore the distinction between public and private variables. Given a multivariate polynomial with n variables $p(x_1, .., x_n)$ over $GF(2)$ in algebraic normal form (ANF), and a term t_I containing variables from an index subset I that are multiplied together, the polynomial can be written as the sum of terms which are supersets of I and terms that miss at least one variable from I:

$$p(x_1, .., x_n) \equiv t_I \cdot p_{S(I)} + q(x_1, .., x_n)$$

$p_{S(I)}$ is called the *superpoly* of I in p. Note that the superpoly of I in p is a polynomial that does not contain any common variable with t_I, and each term in $q(x_1, .., x_n)$ does not contain at least one variable from I.

For example, consider the polynomial of degree 3 in 5 variables

$$p(x_1, x_2, x_3, x_4, x_5) = x_1x_2x_3 + x_1x_2x_4 + x_2x_4x_5 + x_1x_2 + x_2 + x_3x_5 + x_5 + 1$$

Let $I = \{1, 2\}$ be an index subset of size 2. We can represent p as:

$$p(x_1, x_2, x_3, x_4, x_5) = x_1x_2(x_3 + x_4 + 1) + (x_2x_4x_5 + x_3x_5 + x_2 + x_5 + 1)$$

$$\begin{aligned} t_I &= x_1x_2 \\ p_{S(I)} &= x_3 + x_4 + 1 \\ q(x_1, x_2, x_3, x_4, x_5) &= x_2x_4x_5 + x_3x_5 + x_2 + x_5 + 1 \end{aligned}$$

The main observation of the cube attack is that the symbolic sum over $GF(2)$ of all the derived polynomials obtained from the master polynomial $p(x_1, .., x_n)$ by assigning all the possible 0/1 values to the subset of variables in the term t_I is exactly $p_{S(I)}$ which is the superpoly of t_I in $p(x_1, .., x_n)$. For example, consider the sum of the four polynomials derived from the master polynomial $p(x_1, x_2, x_3, x_4, x_5)$ defined above, by assigning all four possible values of x_1 and x_2 (which appear in the term $t_I = x_1x_2$). The result of this summation is the superpoly of t_I, $p_{S(I)} = (x_3 + x_4 + 1)$. A *maxterm* of p is a term t_I such that the superpoly of I in p is a linear polynomial which is not a constant.

The cube attack has two phases: the preprocessing phase, and the online phase. The preprocessing phase is not key-dependant and is performed once per cryptosystem. In this phase, the attacker finds sufficiently many maxterms of the master polynomial. For each maxterm, he computes the coefficients of the secret variables in the symbolic representation of the linear superpoly. The main challenge of the attacker in the preprocessing phase is to find sufficiently many maxterms with linearly independent superpolys. The attacker randomly chooses a subset I of public variables and uses efficient linearity tests to check whether its superpoly is linear. In case the subset I is too small, the superpoly is likely to be nonlinear and the attacker adds a public variable to I and repeats the process. In case I is too large, the sum will be a constant function, and in this case he drops one of the public variables from I and repeats the process. The correct choice of I is the borderline between these cases, and if it does not exist the attacker retries with a different initial I. Once sufficiently many maxterms with linearly independent superpolys are found, the preprocessing is finished.

During the online phase, the secret variables are fixed. The attacker evaluates each linear superpoly by summing over the values of the cryptosystem for every possible assignment to its maxterm. The secret key can then be recovered by simple linear algebra techniques. When the polynomial is assumed to be d-random (i.e., each term has degree at most d and each term of degree exactly d occurs with probability 0.5), it was shown in [9] that almost any choice of $d - 1$ secret variables is a maxterm, and the superpolys of these maxterms are independent random linear combinations of the other variables even when the superpolys are computed from subcubes with large intersections. Consequently, cube attacks can break with high probability any scheme represented by a d-random polynomial in n input variables over $GF(2)$ in about $n2^{d-1} + n^2$ bit operations.

3 Error Correction

The 0/1 value of the leaked information is usually obtained by the attacker through an indirect measurement of an analog value such as power consumption, and is likely to contain errors due to noise and quantization problems. For example, the Hamming weight of a byte written into a register is typically calculated based on the power consumption of the chip. This measurement can be influenced by other sources of power consumption, and therefore it is not

completely accurate. If the attacker wants to determine whether this Hamming weight is larger than or equal to its expected value of 4, then a measurement of 2 or 6 are likely to provide a reliable answer, but a measurement of 4 is too close to the discretization threshold and thus too susceptible to random noise.

In our leakage attacks, we assume that only one bit of information about the Hamming weight at each round is available to the cryptanalyst. The most robust bit is the msb of the Hamming weight, but in the context of cube attacks its multivariate polynomial representation has the highest possible degree. On the other hand, the lsb of the Hamming weight can be represented by a polynomial of degree 1 (it is just the XOR of all the bits of the word), but it is very sensitive to noise. A good compromise might be to consider the two least significant bits of the Hamming weight, using the values of 00 and 10 as reliable representations for 0 and 1, and the values of 01 and 11 as uncertain discretization buffer zones.

A more realistic model of leakage attacks is thus a model in which each leaked bit has three possible values: 0, 1, and $\perp$, where a $\perp$ indicates a problematic measurement which cannot be relied upon. This model is closely related to erasure codes (see [13], for example), in which the recipient of some communication knows which of the received bits are correct and which bits might have been flipped, and uses an appropriate error correcting code in order to overcome the noise.

The original cube attack is extremely sensitive to errors, since it typically sums (modulo 2) millions of 0/1 measurements in order to get the right hand side of a single linear equation, and repeats the summation over multiple overlapping subcubes in order to determine all the equations. In fact, even a negligible fraction of $\perp$ measurements can foil the original cube attack since it makes the numeric values of *all* the subcube sums simultaneously unknown with probability which is exponentially close to 1.

Our main observation is that cube attacks have a natural error correction mechanism, which is based on the fact that if we collect more measurements, we can get additional linear equations by summing over a larger number of possible subcubes. On the other hand, the larger number of measurements introduces more $\perp$ values, and thus we have to analyze which of the two opposing effects is stronger.

Let us denote by ϵ the fraction of the $\perp$ values among all the measurements which are available to the attacker. We now describe a modified cube attack which can handle such errors, and compute the maximal ϵ for which it can recover the key when we assume that the errors are uniformly distributed and that the leakage function is a d-random multivariate polynomial. The attacker chooses a big Boolean cube defined by assigning all the possible 0/1 values to $k \geq d + \lg n$ public variables, and performs preprocessing by computing all the coefficients of all the $\binom{k}{d-1}$ linear equations which are defined by summing over all the possible subcubes of dimension $d-1$ in the big cube of dimension k. During the online phase, the attacker obtains 2^k leaked bits, one for each assignment of 0/1 values to the public variables of the big cube. Out of the 2^k values, $\epsilon \cdot 2^k$ values are $\perp$ due to uncertainty in the measurement of the leakage function. The

attacker assigns a new variable z_i to each one of these unknown values, and sums both the known 0/1 values and the unknown z_i variables over each one of the $\binom{k}{d-1}$ overlapping subcubes of the big cube. The result of each summation is the sum of a subset of the z_i's, plus 0 or 1 (which represents the numeric sum of the known measurements). The attacker equates each linear combination of the z_i's to the corresponding linear combination of key variables k_j calculated during preprocessing, and obtains a system of $\binom{k}{d-1}$ linear equations in the $\epsilon \cdot 2^k + n$ variables z_i and k_j. Since the equations are random looking, the attacker needs slightly more than $\epsilon \cdot 2^k + n$ equations to solve the system. To make this possible, the number of subcubes $\binom{k}{d-1}$ should be larger than $\epsilon \cdot 2^k + n$. Note that $\binom{k}{d-1} \leq \frac{1}{\sqrt{0.5 \cdot \pi \cdot k}} \cdot 2^k$, hence we cannot correct more than $\frac{1}{\sqrt{0.5 \cdot \pi \cdot k}}$ fraction of errors. The number of equations is equal to the number of variables when $k \approxeq 2(d-1)$ (assuming $n \ll \epsilon \cdot 2^k$). The attacker can thus find the key when at most $\frac{1}{\sqrt{\pi \cdot (d-1)}}$ fraction of the leaked bits are $\perp$, and the best choice for the dimension k of the big cube in this case is about $2(d-1)$.

In feasible attacks $k = 2(d-1) < 50$, and thus $\frac{1}{\sqrt{\pi \cdot (d-1)}}$ is bigger than $\frac{1}{\sqrt{\pi \cdot 25}} \approxeq 0.11$. Consequently, the attacker can find the complete key even when 11% of the leaked bits are too noisy to measure accurately. Since d occurs in the denominator, lower degree polynomials can be solved with an even larger fraction of errors. For example, when the leakage is the Hamming weight of a byte, we can consider the boolean function that is equal to 1 if and only if the Hamming weight is greater than or equal to 2. Given that the degree of this function (in terms of the key and plaintext bits) is no more than $d = 26$ and that the measured byte is uniformly distributed, we set the buffer zone of $\perp$ values to include any byte with Hamming weight 2. Since the buffer zone contains only $\frac{28}{256} = 10.9\% < 11\%$ of the 256 possible values of the measured byte, our modified cube attack should be able to handle this case. This should be contrasted with the extreme sensitivity of the original cube attack, in which even a single $\perp$ value made it impossible to compute the right hand sides of all the linear equations whose subcubes contained this error.

When the leakage function is not assumed to be d-random, the error correction technique remains the same, but the number of required equations and the maximal fraction of errors that we can correct may vary depending on the structure of the linear equations, the size and overlap of the various subcubes, etc.

We implemented the attack on d-random polynomials to verify that there are no hidden properties that may foil the attack (such as unexpected linear dependencies between the generated linear equations), and our simulations behaved exactly as expected.

Note that there are many other possible variants of the cube attack which may overcome even more errors when more data is given (for example, the attacker can try to use only subcubes which contain a relatively small subset of errors). However, when the equations are random looking and the errors are randomly distributed, these approaches do not look promising.

4 Serpent

Serpent [2] is a block cipher designed by Ross Anderson, Eli Biham, and Lars Knudsen. It was submitted as a proposal for the Advanced Encryption Standard, and was chosen as one of its five finalists. Serpent is considered to be the AES submission which has the largest security margin, as the best known attack can break only 12 out of its 32 rounds.

Serpent is a substitution-permutation network operating on a block of four 32-bit words, and supports key sizes of 128, 192 or 256 bits. The cipher has 32 rounds, where in each round one of eight possible 4-bit to 4-bit S-boxes is applied 32 times in parallel. Serpent was designed so that all its operations can be executed in parallel, using four 32-bit words to represent the 128 1-bit slices. The best attack on round-reduced variants of Serpent is a differential-linear attack on 10 rounds with a 128-bit key and on 12 rounds with a 256-bit key [14]. Other known attacks include the boomerang [15] and rectangle [16] attacks, which are extensions of differential cryptanalyses. All these attacks cannot be naturally combined with side channel attacks that leak only one state bit in each round.

We first present the best known non-cube leakage attack on Serpent, which was described to us in a private communication with Orr Dunkelman and Nathan Keller (in their forthcoming paper [17], they consider a different situation where the attacker has a much smaller number of available plaintexts; it appears that in this case, even attacking variants with a very small number of rounds is quite complicated). We then present our side channel cube leakage attack which is far more general and much more efficient.

4.1 A Side Channel Linear Attack on Serpent

We assume that we have one known state bit after rounds 1 and 2 of Serpent with index at most 31 (i.e. it has the smallest index among the four bits that are inserted to the first Sbox). Each one of these bits has the property that the inverse linear transformation of Serpent diffuses it to at most three bits. Initially, we use the known bit at the end of the first round to guess and verify all the 12 key bits that it depends on. Then, we consider a 1-round linear approximation in the second round (ending with the single bit we know at the output of the second round), which influences 6 Sboxes (the least number of Sboxes) in the previous round. Then we guess all the 24 key bits of the first round needed to compute the linear approximation and check whether it holds. The bias of the approximation is about $2^2(2^{-3})^3 = 2^{-7}$, so this phase requires 2^{17} known plaintexts and its complexity is about 2^{40}. We then consider more linear approximations that allow us to guess more key bits, and recover most of the key. The total complexity of the attack is about $2^{24} \cdot 2^{17} = 2^{41}$ which can probably be improved (for example, we may already know from the first phase some of the bits that we guess for the linear approximation), but it seems unlikely that improvements will make it faster than the cube attack (described next). Moreover, this attack is specific to the case when the index of the leaked state bit (which is not necessarily

controlled by the attacker) happens to be at most 31, and for other leaked bits the complexity is considerably higher.

4.2 Side Channel Cube Attacks on Serpent

Serpent achieves complete diffusion after 3 rounds, and thus it is not possible to recover the full key given just a single state bit after rounds 1 and 2. Using cubes of dimension 24, we were able to get constant superpolys for any single state bit after 4 Serpent rounds. However, it seems that cubes of a much bigger dimension are required in order to get maxterms for any single state bit after 4 or more rounds. This is not surprising considering the fact that every one of the output bits of Serpent Sboxes can be described as a polynomial of degree 3 in its inputs, and so the degree of the state bit polynomials after 4 rounds of Serpent can theoretically be close to $3^4 = 81$.

Since complete key recovery given a single state bit after rounds 1 or 2 is impossible, and key recovery attacks seem infeasible for 4 rounds and beyond, we concentrate on key recovery attacks given a single state bit at the end of round 3. Our computations show that the degree of any such polynomial in the plaintext and key bits after 3 encryption rounds is at least 10, and thus it requires an infeasible amount of space to even write down these polynomials in an explicit form (a random polynomial of degree 10 in 128 variables has about $\binom{128}{10} \approxeq 2^{48}$ terms). Consequently, we cannot use the standard algebraic attacks to solve a system of such polynomial equations, but we can easily handle them with a cube attack.

During the preprocessing phase of the cube attack, we were able to find 128 maxterms with linearly independent superpolys given just the first bit of the 3rd round encryption of Serpent. This suffices in order to recover all the 128 key bits with trivial complexity, using Gauss elimination. All of the maxterms correspond to surprisingly small cubes of dimension 11, which are listed in Table 1. Each of the maxterms passed at least 100 linearity tests, and thus the maxterm equations are likely to be correct for most keys. During the online phase of the cube attack, the attacker has to find the right hand sides of the linear equations defined by these maxterms by summing over cubes of dimension 11, and thus the complexity of the attack is about $128 \cdot 2^{11} = 2^{18}$. This is much faster than the linear attack described in the previous subsection and has the additional advantage that the choice of the state bit makes almost no difference for the success of the attack, and in fact the attack can be applied even when the attacker does not know which state bit he is getting. Note that the public plaintext variables of the cubes we found and the secret key variables in the resultant linear equations are far from random looking: Variables that appear together as inputs to the first Sbox layer (i.e. their indexes have the same values modulo 32) tend to appear together in the maxterms and also in the linear equations.

5 AES

AES [1] is the Advanced Encryption Standard adopted by NIST following a 5-year competition. The cipher was developed and submitted by Joan Daemen and Vincent Rijmen, and was originally called Rijndael. AES is a substitution-permutation network that supports key sizes of 128, 192 or 256 bits, operating on a block of 128 bits. AES has 10, 12 or 14 rounds (depending on the size of the key), where in each round, the 128-bit state block is viewed as a 4×4 array of 8-bit bytes and undergoes a series of linear and non-linear operations over the field GF(2^8). Currently there is no known algorithm that breaks AES faster than exhaustive search. The best known attack on AES [18] is a variant of the original square attack that was published with the cipher. In this paper we concentrate on side channel attacks on AES with a 128-bit key, although our observations easily extend to AES with a key of 192 or 256 bits.

5.1 A Side Channel Cube Attack on AES

AES reaches complete diffusion after 2 rounds, so a key recovery attack given any single bit at the end of the first round is impossible. Assuming that the leakage function is a single state bit at the end of the second round, we use the known structure of AES to intelligently search for maxterms: After 1 round, the 16 bytes of the AES state can be split into four 4-byte groups that are not mixed with each other. In the second round, these groups are mixed, but only linearly. Thus, if we sum the leaked state bits on a cube defined by plaintext bits that are not all from the same group, we will get a constant value. In addition, according to the square property of AES, if we sum on a cube that contains a Λ-set (as defined in [1]) in which at least one byte is active, we will also get a constant value. Based on these observation, we were able to find after a short search maxterms for the polynomial defined by any state bit at the end of the second round by choosing cubes of dimensions 27 and 28, containing plaintext indexes from the same group, but not containing any Λ-set. Since these relatively low degree polynomials contain all the 128 bits from the initial key, we can use the cube attack to recover all of the 128 key bits of AES, given an arbitrary single bit at the end of the second AES round with complexity of about $128 \cdot 2^{28} = 2^{35}$. However, in this case we also found a non-cube attack which has roughly the same complexity, as described next.

5.2 A non-cube Attack on AES

The attack is based on the square property that states that a Λ-set in which only one byte is active, will result in a complete column of active bytes after 1 round, while the other bytes remain constant. We use the extension of the Square attack by one round in the beginning: We guess 4 bytes of the initial key, and choose a set of 256 plaintexts that results in a Λ-set at the output of the first round with a single active Sbox. The key byte indexes that are guessed and the plaintext values are chosen such that a correct guess of the key will result in

a constant leakage function value after 2 rounds. An incorrect guess of the 4 key bytes is likely to be eliminated quickly, and we remain with the correct guess for these 4 key bytes after about 2^{32} operations. This process can be repeated for 3 of the four 4-byte sets of the initial key. The remaining 4 key bytes can be recovered by exhaustive search. The total complexity of the attack is about 2^{36}.

Based on the square properties for 2 and 3 rounds, the attack can be naturally extended to cases where the leakage function is a single bit at the end of either the third or the fourth round. The complexity of the attacks on 3 and 4 rounds is higher and they require more memory, but they are still completely feasible.

Table 1. Maxterms for 3-round Serpent given the first state bit. Equations are given in the working key bits that are inserted to the first Sbox layer.

Maxterm Equation	Cube Indexes	Maxterm Equation	Cube Indexes
1+x0	{3,8,21,35,46,78,85,96,99,104,117}	x16+x48	{10,25,42,57,62,80,94,106,112,121,126}
x0+x96	{7,13,32,34,45,64,66,77,98,103,109}	1+x16+x112	{6,24,25,38,48,56,57,80,102,120,121}
x32	{7,13,34,45,64,66,77,96,98,103,109}	1+x48	{3,10,13,16,35,45,80,94,99,106,109}
x64	{0,2,8,34,36,40,68,96,98,100,104}	1+x80	{10,11,16,17,48,49,74,75,106,107,113}
x1+x33	{2,3,23,34,35,65,87,97,98,99,119}	x17+x49	{3,14,22,35,54,78,81,99,110,113,118}
x1+x97	{18,19,20,33,51,52,65,82,114,115,116}	x17+x113	{0,22,32,49,54,63,81,86,95,96,127}
1+x33	{1,18,19,20,50,51,52,65,82,115,116}	x49	{0,32,54,63,81,86,95,96,113,118,127}
1+x65	{1,18,19,20,33,50,51,52,82,115,116}	1+x81	{0,17,22,32,49,54,63,86,95,96,127}
1+x2	{5,16,37,48,58,76,90,98,101,112,122}	x18+x50	{10,20,31,42,52,82,95,106,114,116,127}
x2+x34	{4,13,21,36,45,66,85,98,100,109,117}	1+x50+x114	{10,18,20,42,52,63,82,95,106,116,127}
1+x2+x98	{4,13,21,34,36,45,66,85,100,109,117}	x82	{10,18,20,31,42,52,95,106,114,116,127}
x66	{4,13,21,34,36,45,85,98,100,109,117}	1+x114	{13,18,45,53,55,77,85,87,109,117,119}
1+x3	{2,6,13,34,38,45,66,74,99,102,109}	1+x19+x115	{18,21,39,50,51,53,71,83,103,114,117}
x3+x35	{0,22,30,62,64,67,86,96,99,118,126}	x51	{3,4,24,56,67,68,83,99,100,115,120}
1+x35+x99	{0,3,22,30,62,64,67,86,96,118,126}	1+x51+x115	{18,19,21,39,50,53,71,83,103,114,117}
x67	{3,26,27,30,62,90,91,99,122,123,126}	x83	{18,21,39,50,51,53,71,103,114,115,117}
1+x4	{10,13,15,42,45,74,77,79,100,106,111}	1+x20+x116	{12,17,24,44,49,52,56,84,88,108,113}
x36	{0,16,21,32,48,53,68,80,96,100,117}	x52	{6,14,38,46,53,84,85,102,110,116,117}
1+x36+x100	{0,2,4,8,34,40,64,68,96,98,104}	1+x52+x116	{12,17,20,44,49,56,84,88,108,113,120}
1+x68	{0,2,4,8,34,36,40,64,96,98,104}	1+x84	{12,17,20,44,49,52,56,88,108,113,120}
x5+x37	{10,20,31,42,52,69,95,101,106,116,127}	1+x21+x117	{10,17,49,53,54,74,85,86,106,113,118}
1+x5+x101	{2,7,20,34,37,39,52,66,69,103,116}	1+x53	{6,10,21,27,38,59,74,85,102,106,123}
x37	{10,20,31,42,69,74,84,101,106,116,127}	1+x53+x117	{10,17,21,22,49,54,74,85,86,106,113}
1+x69	{5,10,13,16,37,42,45,48,77,106,112}	x85	{4,13,21,34,36,45,66,98,100,109,117}
1+x6	{14,28,29,41,46,60,93,102,110,124,125}	x22+x118	{10,17,49,53,54,74,85,86,106,113,117}
x6+x102	{1,15,16,38,48,65,70,79,97,111,112}	1+x54	{10,17,21,22,49,74,85,86,106,113,117}
x38	{16,24,25,48,56,70,89,102,112,120,121}	1+x54+x118	{0,3,22,30,62,64,67,86,96,99,126}
1+x70	{6,18,23,31,38,50,55,63,82,119,127}	1+x86	{10,17,22,49,53,54,74,85,106,113,117}
1+x7	{13,32,34,45,64,66,77,96,98,103,109}	x23+x55	{10,16,31,42,48,63,74,87,112,119,127}
x7+x39	{18,19,21,50,53,71,83,103,114,115,117}	x55	{3,22,35,40,54,61,72,87,99,118,119}
x71	{18,21,39,50,51,53,83,103,114,115,117}	1+x55+x119	{16,23,31,42,48,63,74,87,106,112,127}
1+x103	{7,13,32,34,45,64,66,77,96,98,109}	x87	{10,16,23,31,42,48,63,74,112,119,127}
x8+x104	{21,31,40,53,54,72,86,95,117,118,127}	1+x24	{7,9,16,29,39,48,103,105,112,120,125}
1+x40	{3,8,14,21,46,53,67,72,78,99,117}	x24+x120	{12,17,20,44,49,56,84,88,108,113,116}
1+x72	{3,8,22,28,35,40,54,60,92,99,118}	x56	{12,17,20,44,49,52,84,88,108,113,120}
1+x72+x104	{3,22,23,35,40,54,55,87,93,99,118}	x88	{24,25,26,28,57,58,60,89,120,122,124}
1+x9	{1,10,33,42,47,65,74,79,105,106,111}	x25+x57	{16,24,38,48,56,70,89,102,112,120,121}
1+x41	{3,9,21,53,59,67,73,91,99,117,123}	1+x57	{22,24,25,41,56,73,86,89,105,118,120}
1+x73	{3,9,21,41,53,59,67,91,99,117,123}	1+x57+x121	{16,24,25,38,48,56,70,89,102,112,120}
1+x105	{1,9,10,15,33,42,47,65,74,79,106}	x89	{16,24,38,48,56,57,70,102,112,120,121}
x10+x42	{12,17,43,44,49,74,75,106,107,108,113}	x26+x122	{0,9,23,32,41,55,58,73,90,96,119}
1+x42+x106	{10,11,12,17,44,49,74,75,107,108,113}	x58	{0,9,23,32,41,55,73,90,96,119,122}
1+x74	{16,23,31,42,48,63,87,106,112,119,127}	1+x58+x122	{2,5,16,26,37,48,76,90,98,101,112}
1+x106	{6,10,13,34,38,45,66,67,98,102,109}	1+x90	{0,9,23,32,41,55,58,73,96,119,122}
1+x11+x107	{10,12,17,42,43,44,49,74,75,108,113}	x27+x59	{3,26,30,62,67,90,91,99,122,123,126}
1+x43	{11,17,22,26,58,75,81,86,113,118,122}	x59	{20,31,52,54,63,86,91,116,118,123,127}
1+x43+x107	{10,11,12,17,44,49,74,75,106,108,113}	1+x59+x123	{3,26,27,30,62,67,90,91,99,122,126}
x75	{10,12,17,42,43,44,49,74,107,108,113}	x91	{3,26,27,30,62,67,90,99,122,123,126}
1+x12	{5,16,26,37,48,58,66,90,101,108,112}	x28+x60	{17,30,32,49,62,64,92,96,113,124,126}
x12+x108	{5,7,9,39,41,44,69,76,101,103,105}	1+x60	{6,25,26,28,38,57,58,89,92,102,122}
1+x44	{5,7,9,12,39,41,69,76,101,103,105}	1+x92	{17,30,32,49,60,62,64,96,113,124,126}
x76	{12,20,27,28,59,84,92,108,116,123,124}	1+x28+x124	{17,30,32,49,60,62,64,92,96,113,126}
x13+x45	{2,12,20,34,52,61,77,98,108,109,116}	x29+x61	{0,8,9,32,40,41,64,93,104,105,125}
x13+x109	{4,10,15,42,45,47,74,77,79,100,106}	x29+x125	{0,6,18,32,38,50,61,93,96,102,114}
x45	{14,20,46,52,60,77,92,109,110,116,124}	1+x61	{0,6,18,29,32,38,50,93,96,102,114}
1+x77	{4,10,15,42,45,47,74,79,100,106,109}	1+x93	{0,6,18,29,32,38,50,61,96,102,114}
x14	{3,27,35,56,64,78,88,99,110,120,123}	1+x30	{4,9,32,36,41,64,73,96,100,105,126}
x14+x46	{15,16,20,47,52,78,80,110,111,112,116}	1+x30+x126	{12,17,24,44,49,56,62,88,94,108,113}
x14+x110	{0,19,23,32,46,51,55,78,87,96,115}	x62	{9,22,24,41,56,73,86,94,118,120,126}
1+x78	{0,14,19,32,46,51,55,87,96,115,119}	1+x94	{9,22,24,41,56,62,73,86,118,120,126}
x15+x111	{3,10,20,35,42,47,52,67,79,106,116}	x31+x63	{10,18,20,42,50,52,82,95,106,116,127}
x47	{10,16,42,48,63,79,95,106,111,112,127}	1+x31+x127	{10,18,20,42,52,63,82,95,106,114,116}
1+x47+x111	{9,10,15,33,42,65,74,79,97,105,106}	x63	{1,12,17,33,44,49,65,95,108,113,127}
1+x79	{1,9,10,15,33,42,47,65,74,105,106}	x95	{10,18,20,42,52,63,82,106,114,116,127}

6 Conclusions and Open Problems

In this paper we demonstrated that cube attacks are ideal tools in leakage attacks, due to a combination of several factors:

1. Cube attacks can be naturally applied to situations in which only a single bit of information in each encryption is available to the cryptanalyst. Almost all the other cryptanalytic techniques require knowledge of big chunks of data in order to partially encrypt or decrypt them.
2. Leaked bits from early rounds of the block cipher are typically represented by polynomials of very moderate degree, and thus a cube attack can exploit them even when it cannot be applied to the full block cipher due to its very high degree.
3. New side channel attacks are found every few months, but each type of leaked information requires specific analysis. Cube attacks are generic tools which can be automatically applied to any new kind of leakage as soon as it is suggested.
4. In many cases, it is difficult to relate the signal extracted by the side channel attack to the underlying mathematical computation. For example, the EM radiation picked up by a small antenna may be a combination of the contributions from several nearby wires, and the total power consumption of the chip may be a very complex function of several computations which are carried out in parallel. Cube attacks do not require any knowledge of the physical implementation or understanding of how the measured signal is generated: If the signal is a low degree polynomial, the cube attack will detect it and extract the key in an automatic way.
5. Many chips use secret countermeasures against side channel attacks. For example, the bus between the ALU and memory may be protected from probing attacks by scrambling the data and address lines with a chip-specific key. If the simple bus encryption scheme can be described by a low degree polynomial, the cube attack can be applied even when the details of the countermeasures are unknown to the attacker.

There are many problems left open in this area. Here are some of them:

1. How to find the best maxterms of a blackbox multivariate polynomial, and how to convincingly estimate their asymptotic degrees when finding them explicitly is infeasible.
2. How to exploit the structure of a given S-P network in order to concentrate on the most promising types of maxterms in leakage attacks.
3. How to overcome even more noise in the leaked bits by using improved cube attacks.
4. What are the best cube and non-cube leakage attacks against all the major block ciphers.

References

1. Joan Daemen and Vincent Rijmen. AES proposal: Rijndael. Technical Evaluation, CD-1: Documentation, 1998.
2. Ross Anderson, Eli Biham, and Lars Knudsen. Serpent: A proposal for the advanced encryption standard. In *in First Advanced Encryption Standard (AES) Conference*, 1998.
3. Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. In *CRYPTO '90: Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology*, pages 2–21, London, UK, 1991. Springer-Verlag.
4. Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In *EUROCRYPT '93: Workshop on the theory and application of cryptographic techniques on Advances in cryptology*, pages 386–397, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc.
5. Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: the case of AES. In *Topics in Cryptology - CT-RSA 2006, The Cryptographers Track at the RSA Conference 2006*, pages 1–20. Springer-Verlag, 2006.
6. Silvio Micali and Leonid Reyzin. Physically observable cryptography. In *TCC 2004, LNCS*, pages 278–296. Springer, 2004.
7. M. Yung F.-X. Standaert, T.G. Malkin. A unified framework for the analysis of side-channel key recovery attacks. In *EUROCRYPT '09: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques*, pages 443–461. Springer-Verlag, 2009.
8. Alex Biryukov. Design of a new stream cipher–LEX. In *New Stream Cipher Designs: The eSTREAM Finalists*. pages 48–56, 2008.
9. Itai Dinur and Adi Shamir. Cube attacks on tweakable black box polynomials. In *EUROCRYPT '09: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques*. Springer-Verlag, 2009.
10. Michael Vielhaber. Breaking ONE.FIVIUM by AIDA an algebraic IV differential attack. Cryptology ePrint Archive, Report 2007/413, 2007.
11. Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. *Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations*. 2000.
12. Jean Charles Faugère. A new efficient algorithm for computing gröbner bases without reduction to zero (f5). In *ISSAC '02: Proceedings of the 2002 international symposium on Symbolic and algebraic computation*, pages 75–83, New York, NY, USA, 2002. ACM.
13. M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman. Efficient erasure correcting codes. *Information Theory, IEEE Transactions on*, 47(2):569–584, 2001.
14. Orr Dunkelman, Sebastiaan Indesteege, and Nathan Keller. A differential-linear attack on 12-round serpent. In *INDOCRYPT '08: Proceedings of the 9th International Conference on Cryptology in India*, pages 308–321, Berlin, Heidelberg, 2008. Springer-Verlag.
15. John Kelsey, Tadayoshi Kohno, and Bruce Schneier. Amplified boomerang attacks against reduced-round mars and serpent. In *FSE '00: Proceedings of the 7th International Workshop on Fast Software Encryption*, pages 75–93, London, UK, 2001. Springer-Verlag.
16. Eli Biham, Orr Dunkelman, and Nathan Keller. The rectangle attack - rectangling the serpent. In *EUROCRYPT '01: Proceedings of the International Conference on*

the Theory and Application of Cryptographic Techniques, pages 340–357, London, UK, 2001. Springer-Verlag.

17. Orr Dunkelman and Nathan Keller. Low data complexity attacks on reduced-round AES (in preparation).
18. Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Michael Stay, David Wagner, and Doug Whiting. Improved cryptanalysis of rijndael. In *FSE '00: Proceedings of the 7th International Workshop on Fast Software Encryption*, pages 213–230, London, UK, 2001. Springer-Verlag.