Using Tools from Error Correcting Theory in Linear Cryptanalysis

Benoît GÉRARD, Jean-Pierre TILLICH

INRIA Paris-Rocquencourt, Project-Team SECRET, B.P.105 Le Chesnay Cedex 78153, France.

Abstract. The main topics where coding theory can be useful for linear cryptanalysis are finding linear approximations of a cipher [FLT09a, FLT09b], recovering efficiently key bits from statistical data [BCQ04, GT09] and estimating the data complexity of a linear cryptanalysis [GT09]. We give an overview of such results here.

1. Introduction

Linear cryptanalysis is probably one of the most powerful tools available for attacking symmetric cryptosystems. It was invented by Matsui [Mat94,Mat94] to break the DES cipher building upon ideas put forward in [TCG91,MM92]. It was quickly discovered that other ciphers can be attacked in this way, for instance FEAL [OA94], LOKI [TSM94], SAFER [MPWW95].

It is a known plaintext attack which takes advantage of probabilistic linear equations that involve bits of the plaintext $\mathbf{p}$, the ciphertext $\mathbf{c}$ and the key $\mathbf{k}$ of the kind

$$\Pr(\pi \cdot \mathbf{p} \oplus \gamma \cdot \mathbf{c} = 0)\frac{1}{2} + (-1)^{\kappa \cdot \mathbf{k}}\varepsilon. \tag{1}$$

ε is called the *bias* of the equation, π, γ and κ are linear masks and $\pi \cdot \mathbf{p}$ denotes the following inner product $\pi \cdot \mathbf{p} \overset{\text{def}}{=} \bigoplus_{i=1}^{m} \pi_i p_i$ between $\pi = (\pi_i)_{1 \leq i \leq m}$ (where m denotes the message size) and $\mathbf{p} = (p_i)_{1 \leq i \leq m}$. There might be several different linear approximations of this kind at our disposal and we let n be their number. We denote the corresponding key masks by κ_i and the corresponding biases by ϵ_i for $i \in \{1, \ldots, n\}$. Another important quantity which will appear repeatedly is the dimension d of the vector space generated by the κ_i's.

There have been several papers during the last years exploiting strong links between linear cryptanalysis and coding theory to obtain new results on the first topic. The results obtained by this approach are of several kinds:

- finding good linear approximations can be treated by using efficient algorithms for decoding first order Reed Muller codes;

- the information available for the attacker by using the linear approximations on the set of available plaintext/ciphertext pairs can be viewed as obtaining the result of the transmission of several bits of a linear code on a Gaussian channel, this can be used to bring in linear decoding techniques to speed up the recovery of the key bits once the relevant statistical data is computed;
- information theoretic tools can be used to quantify with accuracy the amount of plaintext/ciphertext pairs which are needed in several different linear cryptanalyses.

In Section 2, we review the results of [FLT09a,FLT09b,RT09] which deal with using decoding algorithms of the first order Reed Muller codes to find good linear approximations. This algorithm which requires to choose a given output mask, outputs suitable input masks and key masks when they exist. It has complexity $O\left(\frac{v}{\epsilon^2}\right)$, where v is the sum of the input message size m and the key size k, and ϵ is the bias of the linear approximation. This is only slightly worse than testing whether a certain linear approximation has bias of order ϵ. This algorithm can be applied to any cipher (i.e. not just ciphers based on iterated round functions) and does not rely on the piling up lemma approximation. Moreover its complexity is optimal in the black box model where the linear combination of the cipher output $\gamma \cdot \mathbf{c}$ which is chosen is given by a black box and any query to the ciphering function has constant cost. Its complexity is of course worse for small biases when we have a cipher based on iterating several times a given round function and where the piling up lemma approximation is correct. It is for instance right now still a little bit too complex to be applied to a full-round DES. However, it proved to be quite useful for certain side channel attacks [RT09] where all linear masks of the input could exhaustively be checked by the algorithm.

We will also give another application of decoding techniques, this time for improving the complexity of recovering the right key (or subkey) once the relevant statistics associated to the n linear approximations have been computed. The link with error correction techniques comes here from the fact that the statistics available for the attacker can be viewed as the result of the transmission of the linear combinations of the key bits $\kappa_i \cdot \mathbf{k}$ through a Gaussian channel. The naive approach of performing this task would have complexity $\Omega(n2^d)$. This can be easily reduced to $O(d2^d)$ by a fast Fourier transform and up to $O(2^{\frac{d(n-d)}{n}})$ with decoding techniques based on ellipsoids [Dum00]. We present here another decoding algorithm for performing this task whose complexity has never been analyzed rigorously but which is simple to implement and is one of the fastest in practice.

Finally, we also point out that information theory can also be used to give a rather sharp bound on the number N of plaintext/ciphertext pairs which are needed for performing linear cryptanalysis. We depart here from the usual notions of advantage [Sel08,HCN09b] and gain [BCQ04] and quantify here the entropy of the key given the statistical data at hand.There are several reasons for this:

- First, this notion is probably the most relevant quantity to quantify the amount of random bits left in the key knowing the statistics deduced from the available data. It is definitely relevant for linear cryptanalysis: for instance a good rule of thumb to run linear cryptanalysis is to take a list size of order 2^h, where h is the conditional entropy of the key given the statistical data (see [GT09]).

- Second, the gain is in general not an accurate tool for this purpose. This comes from the fact that we are precisely in a setting where the median and the expected value of the rank of the right key differ significantly. This is elaborated on in Section 5.
- Finally, there is a very handy tool for lower bounding this quantity which is at the same very general, easy to apply and surprisingly sharp, namely Theorem 1 which is a well known result in information theory. Even if the amount N of data which is needed for linear cryptanalysis is now well understood [Jun01,Sel08] in the case of one linear approximation, this is not really the case when many linear approximations are available. The results given in [BCQ04] are quite pessimistic as been observed in [GT09]. In the more general setting of multiple linear cryptanalysis it is by no means obvious to have an accurate estimate of the amount N which is needed for a given list size and success probability. In this case, having a handy tool for estimating the entropy of the key together with the aforementioned rule of thumb is quite useful. It should be mentioned that information theoretic tools can also be brought in to prove that this rule of thumb is indeed correct in certain situations but this is beyond the scope of this paper.

2. Finding linear approximations using a decoding algorithm for order 1 Reed-Muller code

In this section we explain how to find good linear approximations for a Boolean function F, by making as few queries to the function as possible. Basically we provide here an algorithm for performing this task which has time complexity of order $O\left(\frac{v}{\epsilon^2}\right)$, where v is the number of variables of the Boolean function and ϵ is the bias of the linear approximations which is sought (that is the linear approximation and the Boolean function agree on a fraction of inputs which is $\frac{1}{2}+\epsilon$). The complexity of this algorithm is essentially optimal in the black box model (where F is given by a black box and where any query to the black box has complexity $\Theta(1)$). Indeed, we show in this section that *any* algorithm performing this task has to use at least $\Omega\left(\frac{v}{\epsilon^2}\right)$ entries of F. It can be noticed that the time complexity of the algorithm depends only weakly in the number of variables of the Boolean function and that it has about the same time complexity as the apparently much simpler task which is testing whether a certain linear approximation has a bias of order ϵ: any algorithm of this kind must use at least $\Omega\left(\frac{1}{\epsilon^2}\right)$ entries of F.

This algorithm can be used in the linear cryptanalysis context to check whether a certain linear combination of the output bits of a cipher can be well approximated by a certain linear combination of the key bits and the message input bits. When considering ciphers, there exists other techniques that provide good estimates of the biases of linear approximations with better time complexity when the cipher is an iterated cipher by using the piling-up lemma. Nevertheless, the algorithm presented here is of great interest in the scope of side-channel attacks. The attack proposed in [RT09] is equivalent to a linear cryptanalysis using several approximations where the ciphertext is replaced by an intermediate value $I(\mathbf{p}, \mathbf{k})$ of the cipher. More precisely, in the Hamming weight (or distance) model, the information that can be obtained about this value is a power consumption trace that is correlated with the Hamming weight $|I(\mathbf{p}, \mathbf{k})|$ of this register. Hence, each measurement of a trace gives to the attacker the corresponding value of $|I(\mathbf{p}, \mathbf{k})|$. Linear approximations of the form $\mathbf{p} \cdot \pi \oplus \mathbf{c} \cdot \gamma = \mathbf{k} \cdot \kappa$ are thus replaced by $\mathbf{p} \cdot \pi \oplus |I(\mathbf{p}, \mathbf{k})| \cdot \gamma = \mathbf{k} \cdot \kappa$. Then, the attack is performed as for any other non-distinguishing linear cryptanalyses using many approximations. In these attacks, the data complexity is linked to the biases of the approximations (the greater they are, the smaller the complexity is). That is the reason why finding good approximations of the boolean function

$$F : (\mathbf{p}, \mathbf{k}) \mapsto |I(\mathbf{p}, \mathbf{k})| \cdot \gamma,$$

is of interest here. Notice that in order to be sure to get the best approximations, one has to run the algorithm with all possible values for the output mask γ. Here this is not a real problem since the number of bits for $|I(\mathbf{p}, \mathbf{k})|$ is of course relatively small. In some contexts, the attacker may not know which cipher is used and reverse engineering may be costly. In such a case, having a twin device allows the attacker to consider F as a black box and thus to find linear approximations using the algorithm presented here.

Now that the motivation has been exposed, let us focus on the core of the problem. First, we will see that a linear approximation of a Boolean function can be viewed as a decoding problem of Reed-Muller codes of order 1. The problem is that for cryptanalytic issues, codewords are so large that we are not even able to store one of them in computer memory. Then, we will present the probabilistic algorithm provided in [FLT09a,FLT09b] that decodes large length order 1 Reed-Muller codes which solves this issue and uses only a tiny fraction of the codeword. Finally, we will discuss the time and space complexities of this algorithm.

2.1. Reed-Muller codes of order 1

The order 1 Reed-Muller codes are related to linear cryptanalysis via Boolean functions. Let us recall that these codes are defined as follows.

Definition 1 *A Boolean function g in v variables is a function from $\mathbb{F}_2^v$ to $\mathbb{F}_2$. The truth table of a Boolean function g is the vector composed of its evaluations: $(g(\mathbf{u}))_{\mathbf{u} \in \mathbb{F}_2^v}$. The Algebraic Normal Form (ANF) of a Boolean function g is the unique polynomial in $\mathbb{F}_2[x_1, \ldots, x_v]$ of the form $f(x_1, \ldots, x_v) = \sum_{J \subset \{1,\ldots,v\}} a_J \Pi_{j \in J} x_j$ such that*

$$\forall \mathbf{a} \in \mathbb{F}_2^v, g(\mathbf{a}) = f(a_1, \ldots, a_v).$$

The degree of a Boolean function is the degree of its ANF.

Now we can define Reed-Muller codes in terms of Boolean functions.

Definition 2 *The order o Reed-Muller code of length 2^v is the set of truth tables of Boolean functions in v variables of degree less than or equal to o.*

When considering linear cryptanalysis, we are interested in order 1 Reed-Muller codes. Such codes are composed of the truth tables of all affine functions in a given number of variables. Hence, finding the closest codeword to a given vector $\mathbf{y}$ (in terms of Hamming distance) is finding the best affine approximation of the Boolean function with truth table $\mathbf{y}$. This is the approach followed in [FLT09a].

There are two main obstacles with this approach when applied to linear cryptanalysis.

- A cipher $E : (\mathbf{x}, \mathbf{k}) \mapsto \mathbf{y}$ is not a Boolean function.
- The size of the truth tables considered by this approach. Indeed, the number of variables is the size of the input plus the size of the key. Nowadays, the smallest parameters would be about 64 bits for the input and 80 bits for the key, what leads to a minimum of 144 variables. In other words, it is impossible in this case to consider the whole truth table of the Boolean function which would have 2^{144} entries.

The first point implies that in order to find affine approximations using this Reed-Muller code approach, the output mask γ has to be fixed, i.e. we consider the Boolean function $g : (\mathbf{x}, \mathbf{k}) \mapsto E(\mathbf{x}, \mathbf{k}) \cdot \gamma$. The necessity of fixing an output mask is not the main problem since for many ciphers it is easy to guess the form of such good masks. Once an output mask is fixed, the second point remains, finding affine approximations of the cipher amounts to decoding a vector obtained by computing the values $E(\mathbf{x}, \mathbf{k}) \cdot \gamma$ for all possible $\mathbf{x}$ and $\mathbf{k}$'s, that is a vector of length 2^{144} obtained by 2^{144} encryptions in the example at hand. It is clear that classical decoding algorithms for Reed-Muller codes are not suitable for this case. The solution is a probabilistic algorithm presented in [FLT09a] that reconstructs the best affine approximation by computing only a tiny fraction of the received vector.

2.2. Reconstruction algorithm

The probabilistic solution proposed in [FLT09a] is based on the following observation.

Let g be a Boolean function with v variables and π an input mask that provides a good linear approximation of this Boolean function. Let $I \subset \{1, 2, \ldots, v\}$, π_I (resp. $\pi_{\bar{I}}$) be the restriction of π to its indices that belong to I (resp. that do not belong to I). Using the same notation for $\mathbf{x}$: $g(\mathbf{x}) = g(\mathbf{x}_I, \mathbf{x}_{\bar{I}})$, then, for a fixed value $\mathbf{x}_{\bar{I}}$, $\mathbf{X}_I \cdot \pi_I$ is a good linear approximation of the Boolean function $f(\mathbf{X}_I) = g(\mathbf{X}_I, \mathbf{x}_{\bar{I}})$ [1].

Hence, the best linear approximation can be reconstructed variable per variable. This is the main idea underlying Algorithm 1. For simplicity, we denote by $X_1, \ldots, X_v$ the random variables corresponding to the plaintext/key bits. An affine approximation corresponds to a polynomial of degree at most 1 in

[1]While this statement is true for a random Boolean function, it may not be true anymore for a cipher (think about the Hull Effect). We will discuss this point at the end of the subsection.

$p \in \mathbb{F}_2[X_1, \ldots, X_v]$. The algorithm deals with a list $\mathcal{L}$ of possible (truncated) affine approximations. We denote for a given list $\mathcal{L}$ and a polynomial q in $\mathbb{F}_2[X_1, \ldots, X_v]$, by $\mathcal{L} + q$ the following list

$$\mathcal{L} + q \overset{\text{def}}{=} \{p + q; p \in \mathcal{L}\}$$

To begin with, we will present a simplified version of the algorithm presented in [FLT09a] that works for random Boolean functions (and ciphers with no Hull Effect).

Algorithm 1 Reconstruction of linear approximations.

Parameters:

- ε the minimum bias of approximations what we want to keep,
- T the number of samples to generate for the estimation of the bias,
- a rejection parameter $c < 1$.

Input: A Boolean function $g : \mathbb{F}_2^v \to \mathbb{F}_2$ to approximate.
Output : A list $\mathcal{L}$ of affine approximations of g of bias of order $\Omega(\varepsilon)$.

```
1: $\mathcal{L} \leftarrow \{0, 1\}$;
2: $I \leftarrow \emptyset$;
3: for $1 \leq i \leq v$ do
4:     $\mathcal{L} \leftarrow \mathcal{L} \cup (\mathcal{L} + X_i)$;
5:     $I \leftarrow I \cup \{i\}$;
6:     for $p \in \mathcal{L}$ do
7:         $t \leftarrow 0$;
8:         Randomly choose an $\mathbf{x}_{\bar{I}}$;
9:         for $1 \leq j \leq T$ do
10:            Randomly pick a value $\mathbf{x}_I$ in $\mathbb{F}_2^i$;
11:            $t \leftarrow t + p(\mathbf{x}_I) \oplus g(\mathbf{x}_I, \mathbf{x}_{\bar{I}})$;
12:        end for
13:        if $\frac{t}{T} \geq \frac{1}{2} - c\varepsilon$ then
14:            $\mathcal{L} \leftarrow \mathcal{L} \backslash \{p\}$;
15:        end if
16:    end for
17: end for
18: return $\mathcal{L}$;
```

The first point to discuss about this algorithm is its applicability to ciphers. Actually, since only one $x_{\bar{I}}$ is chosen for each iteration, it is possible for a good approximation to be deleted due to the Hull Effect. Indeed, some bits of $x_{\bar{I}}$ may correspond to bits of the key and the randomly chosen value for these bits can be those of a key for which the approximation has a very small bias. The solution to this problem is to take several values for $x_{\bar{I}}$ as proposed in the original algorithm from [FLT09a]. The number of values to take in order to bypass this problem is only of order $\theta(1)$ and thus this improvement does not increase so much the complexity of the algorithm. Then, there are two ways of deciding whether the approximation has to be kept or not. The first is the one proposed in [FLT09a] that consists in summing the counters obtained for these different values of $x_{\bar{I}}$ and to compare the sum to a fixed threshold. The other way of doing this is to individually compare these counters to the threshold and to delete the approximation if none fulfills the condition. In the first case, the rejection parameter c has to be significantly larger than the one taken in the second approach.

The second point of this discussion is T. The required number of samples to detect a bias ε is known to be of order $\Omega(\varepsilon^{-2})$. For sets I with small cardinality, it is impossible to generate that many different samples since we only have $2^{\#I}$ possible values for x_I and thus many unbiased approximations may be kept until $\#I$ reaches $\log_2(\varepsilon^{-2})$. This is a matter of concern since the time complexity of the algorithm heavily depends on the size of $\mathcal{L}$. Actually, this size is growing exponentially at the beginning of the algorithm and then decreases to 1 till the end of the process. The point where the size of the list decreases to 1 is given in [HKL03]. For a value m of $\#I$, the probability of the list to be reduced to a singleton is upper-bounded by $2^{-2^m\varepsilon^2}$. This induces that for $\#I \geq \log_2(\varepsilon^{-2})$, the size of the list is likely to be one. This remark is the basis of the improvement presented in the next subsection.

2.3. Improving the algorithm

In order to bypass the first iterations and to directly begin the algorithm with a small size of list, the idea is to perform a classical decoding of the code restricted to the first variables. The v iterations are thus divided in $v = v_1 + v_2$. After a classical decoding on the v_1 first variables, Algorithm 1 is used to reconstruct the v_2 others.

The decoding of the v_1 variables can be efficiently performed using the Fast Walsh Transform Algorithm. We recall that the Walsh transformed of a real valued vector $\mathbf{y} = (y_{\mathbf{a}})_{\mathbf{a}\in\mathbb{F}_2^{v_1}}$ is

$$\hat{y}_{\mathbf{x}} \overset{\text{def}}{=} \sum_{\mathbf{a}\in\mathbb{F}_2^{v_1}} (-1)^{\mathbf{x}\cdot\mathbf{a}} y_{\mathbf{a}}.$$

The Walsh coefficient $\hat{y}_{\mathbf{x}}$ when applied to $y_{\mathbf{a}} \overset{\text{def}}{=} (-1)^{g(\mathbf{a},\mathbf{x}_{v_2})}$ (where $\mathbf{x}_{v_2}$ denotes a certain vector in $\mathbb{F}_2^{v_2}$) is related to the distance Δ between the truth table of the function $\mathbf{u} \mapsto g(\mathbf{u}, \mathbf{x}_{v_2})$ and the truth table of the linear function $\mathbf{u} \mapsto \mathbf{u}\cdot\mathbf{x}$, by

$$\hat{y}_{\mathbf{x}} = 2^{v_1} - 2\Delta$$

Therefore the distances between all linear functions $\mathbf{u} \mapsto \mathbf{u}\cdot\mathbf{x}$ and the function $\mathbf{u} \mapsto g(\mathbf{u}, \mathbf{x}_{v_2})$ can be computed with time complexity $\Theta(v_1 2^{v_1})$ and memory complexity $\Theta(2^{v_1})$. The resulting improved algorithm for finding linear approximations is given in Algorithm 2.

The ideal value for v_1 is $\log_2(\varepsilon^{-2})$ (as mentioned before) for which the list has a size of 1 with good probability. Nevertheless, this step may be impossible to run with $v_1 \approx \log_2(\varepsilon^{-2})$ due to memory complexities. In that case, on has to take v_1 as large as possible and to run the reconstruction algorithm with a large list for the first iterations.

Algorithm 2 Reconstruction algorithm using FFT.

Parameters:

- ε the minimum bias of approximations what we want to keep,
- T the number of samples to generate for the estimation of the bias,
- a rejection parameter $c < 1$.
- v_1,v_2 such that $v_1 + v_2 = v$.

Input: A Boolean function $g : \mathbb{F}_2^v \to \mathbb{F}_2$ to approximate.
Output : A list $\mathcal{L}$ of affine approximations of g of bias of order $\Omega(\varepsilon)$.

```
1: for $\mathbf{a} \in \mathbb{F}_2^{v_1}$ do
2:     $y_{\mathbf{a}} \leftarrow (-1)^{g(\mathbf{a},\mathbf{x}_{v_2})}$;
3: end for
4: Compute $\hat{\mathbf{y}}$ using the FFT algorithm;
5: $\mathcal{L} \leftarrow \emptyset$;
6: for $\mathbf{a} \in \mathbb{F}_2^{v_1}$ do
7:     if $\frac{\hat{y}_{\mathbf{a}}}{2^{v_1}} - \frac{1}{2} \geq 2c\varepsilon$ then
8:         $\mathcal{L} \leftarrow \mathcal{L} \cup \{\sum_{i=1}^{v_1} a_i X^i\}$;
9:     end if
10:    if $\frac{\hat{y}_{\mathbf{a}}}{2^{v_1}} - \frac{1}{2} \leq -2c\varepsilon$ then
11:        $\mathcal{L} \leftarrow \mathcal{L} \cup \{1 + \sum_{i=1}^{v_1} a_i X^i\}$;
12:    end if
13: end for
14: $I \leftarrow \{1, 2, \ldots, v_1\}$;
15: perform now Algorithm 1 starting directly at step 3 with the value $i = v_1 + 1$.
```

2.4. Complexities and use of the algorithm

If v_1 can be chosen close to $\log_2(\varepsilon^{-2})$, then this algorithm has the following complexities.

- Time complexity: $O\left(\frac{v}{\varepsilon^2}\right)$.
- Memory complexity: $O\left(\frac{1}{\varepsilon^2}\right)$.

It should be noted that this complexity is optimal in the black-box model: the Boolean function we want to approximate is given by a black box and we can make queries to the black box by asking the corresponding output of the Boolean function for a certain entry. We are also allowed to choose the next query according to the results of the queries which have already been made. Each query has an algorithmic cost of $\Theta(1)$. In this model, it is straightforward to check by information theoretic arguments that the number of queries which have to be made to the black-box is of order $\frac{v}{\epsilon^2}$. To verify this claim, we can think of the following game between a sender and a receiver. The sender chooses a random element $\mathbf{a}$ of $\mathbb{F}_2^v$. Then he chooses a random Boolean function F as follows. He chooses the output corresponding to an entry $\mathbf{x}$ as $\mathbf{a}\cdot\mathbf{x}$ with probability $\frac{1}{2} + \epsilon$ and $1 \oplus \mathbf{a}\cdot\mathbf{x}$ otherwise. The receiver has at his disposal the black box corresponding to the function F and an algorithm which computes the best affine approximation of F. Let us say that he is able to find the best affine approximation of F (which is the linear function $\mathbf{x} \mapsto \mathbf{a}\cdot\mathbf{x}$ with very high probability) with n queries to the black box. This would imply that we could send v bits of information through a binary symmetric channel of crossover probability $\frac{1}{2} - \epsilon$ with vanishing error probability (as v goes to infinity) by sending only n bits through the channel. Recall here that such a channel acts as follows: the bits are sent reliably with probability $\frac{1}{2} + \epsilon$ and get flipped with probability $\frac{1}{2} - \epsilon$ independently of each other. The feedback capacity of this channel (that is the capacity of the binary symmetric channel when we are allowed at the sender side to choose the next bit which has to be transmitted by using the knowledge of the bits which have been received at the receiver side, see [CT91][Sec. 8.12], which is the relevant notion when we can make adaptable calls to the black box such as here) is equal to the capacity of the same channel without feedback [CT91][Theorem 8.12.1]. Therefore it is equal to $1 - h\left(1/2 - \epsilon\right) = \theta(\epsilon^2)$, where $h(x) \overset{\text{def}}{=} -x \log_2 x - (1-x)\log_2(1-x)$. This implies that any transmission scheme of this kind should verify (asymptotically in v)

$$\frac{v}{n} \leq 1 - h\left(1/2 - \epsilon\right) = \theta(\epsilon^2),$$

from which it follows that

$$n = \Omega\left(\frac{v}{\epsilon^2}\right).$$

As noticed before, this complexity is only slightly larger than the complexity of checking whether a certain affine approximation has a bias of order ϵ. However, despite of this, this algorithm is still too complex for finding linear approximations of recent ciphers. In this case, the only tools at hand make use of the iterated structure of the cipher and approximations based on the piling up lemma. Nevertheless, this algorithm does not require the knowledge of the function to approximate and only uses this one as a black box. This is the main advantage of this algorithm that has shown its usefulness for certain side-channel attacks [RT09] as mentioned earlier. Indeed, the number of output bits is the size of a register and thus the number of possible output masks is quite small so that all those masks can be exhaustively tested. Moreover, the biases considered are large enough for the algorithm to have a reasonable complexity.

3. Multiple linear cryptanalysis and decoding linear codes over the Gaussian channel

As mentioned in [BCQ04], the problem of multiple linear cryptanalysis can be viewed as a decoding problem over a noisy channel. This remark is the starting point of the work presented in [GT09] where performing multiple linear cryptanalysis can be treated as a linear decoding problem over a binary white additive Gaussian noise channel.

After a short definition of the Gaussian channel, we present the framework proposed in [GT09] that is based on the fact that a certain function of the counters used in linear cryptanalysis behaves like the output of such a channel. We also recall several relevant notions of *maximum likelihood* and *list* decoding.

3.1. Binary additive white Gaussian noise channel

For the first version of linear cryptanalysis (namely Algorithm 1 of [Mat94]), the attacker has access to N pairs $(\mathbf{p_i}, \mathbf{c_i})$ enciphered with the same key $\mathbf{k}$ and computes the number T of pairs such that $\mathbf{p} \cdot \pi \oplus \mathbf{c} \cdot \gamma = 0$. Writing the linear approximation as

$$\Pr[\mathbf{p} \cdot \pi \oplus \mathbf{c} \cdot \gamma = 0] = \frac{1}{2} + (-1)^{\mathbf{k}\cdot\kappa}\varepsilon,$$

we deduce that the counter T follows a binomial distribution of parameters N and $\frac{1}{2}+(-1)^{\mathbf{k}\cdot\kappa}\varepsilon$. ε is quite small here, therefore this distribution can be approximated by a Gaussian distribution $\mathcal{N}\left(\frac{N}{2} + (-1)^{\mathbf{k}\cdot\kappa}N\varepsilon, N\left(\frac{1}{4} - \varepsilon^2\right)\right)$. If we bring in the following affine function of T

$$Y \stackrel{\text{def}}{=} \frac{2T - N}{2N\varepsilon}.$$

then we immediately obtain that Y is normally distributed with an expected value $(-1)^{\mathbf{k}\cdot\kappa}$ and variance $\frac{1-4\varepsilon^2}{4N\varepsilon^2}$. For simplicity, and since $\varepsilon \ll 1$, we will now suppose that Y follows the normal distribution $\mathcal{N}((-1)^{\mathbf{k}\cdot\kappa}, 1/4N\varepsilon^2)$.

Let us now bring in the definition of a Gaussian channel

Definition 3 (Binary Additive White Gaussian noise channel) *Let Y be the random variable corresponding to the output of a Binary Additive White Gaussian Noise (BAWGN) channel of variance σ^2 and let $X \in \{0,1\}$ be the related input variable. Then,*

$$Y = (-1)^X + Z,$$

where Z is a centered normally distributed variable with variance σ^2.

The link between linear cryptanalysis and the Gaussian channel is now obvious. Matsui's Algorithm 1 that recovers one bit of information about the key $\mathbf{k} \cdot \kappa$ can be seen as a process that guesses the input value $(-1)^X = (-1)^{\mathbf{k}\cdot\kappa}$ that has been sent over a BAWGN channel with variance $\sigma^2 = \frac{1}{4N\varepsilon^2}$ thanks to the knowledge of the output Y. The optimal way of guessing the value of X is

$$(-1)^X = \begin{cases} 1 & \text{if } Y > 0, \\ -1 & \text{otherwise.} \end{cases}$$

This is equivalent to guess that

$$\mathbf{k} \cdot \kappa = \begin{cases} 0 & \text{if } T > N/2, \\ 1 & \text{otherwise.} \end{cases}$$

In the case of multiple linear cryptanalysis, the attacker has n linear approximations with plaintext masks γ_i, ciphertext masks π_i and key masks κ_i with biases ϵ_i. The computation of n counters gives him n variables Y_i corresponding to $(-1)^{\mathbf{k}\cdot\kappa_\mathbf{i}} + Z_i$ and from this, his task is to recover the values of $\mathbf{k}\cdot\kappa_\mathbf{i}$. Notice that the Z_i's are not identically distributed since σ_i depends on the bias ε_i of the i-th approximation:

$$\sigma_i^2 \stackrel{\text{def}}{=} \frac{1}{4N\varepsilon_i^2}.$$

We may notice that *if the κ's are not linearly independent, the attacker should not recover each X_i independently.* This can be illustrated by the following simple example. Suppose that we get three linear approximations with respective key masks κ_1, κ_2 and $\kappa_3 = \kappa_1 \oplus \kappa_2$. Then, recovering each bit independently could lead to guess that $\mathbf{k}\cdot\kappa_\mathbf{1} = \mathbf{k}\cdot\kappa_\mathbf{2} = \mathbf{k}\cdot\kappa_\mathbf{3} = 1$, what is impossible since the scalar product is linear. This problem arises as soon as the dimension of the vector space $\mathcal{K}$ spanned by the n key masks κ_i (say d) is smaller than n. This situation is precisely the setting of error-correcting codes. The vector $(\mathbf{k}\cdot\kappa_\mathbf{1}, \dots, \mathbf{k}\cdot\kappa_\mathbf{n})$ can be viewed as the vector obtained by encoding d bits of information about the key using the linear code defined by the key masks

$$\mathcal{C}_{\kappa_1,\dots,\kappa_n} \stackrel{\text{def}}{=} \left\{(\mathbf{k}\cdot\kappa_i)_{1\le i\le n}; \mathbf{k}\in\mathbb{F}_2^d\right\}.$$

Hence, recovering $\mathbf{k}$ from $\mathbf{Y}$ is equivalent to decoding $\mathbf{Y}$ as a codeword of $\mathcal{C}_{\kappa_1,\dots,\kappa_n}$ that has passed through the particular BAWGN channel that has noise variance σ_i^2 at position i. We will assume that the Gaussian random variables Z_i's which affect each coordinate $\mathbf{k}\cdot\kappa_i$ are *independent*, which means that the channel is *memoryless.*

On the memoryless property of the channel. While in the multidimensional approach [HCN09b,HCN08], this independence of the Z_i's is not a matter of concern, the results obtained in [BCQ04,GT09] rely on this hypothesis. In [Mur06], the framework of [BCQ04] is questioned and it turns out that it can be proved that the vector $(Z_i)_{1\le i\le n}$ is a Gaussian vector. The dependencies between counters are therefore entirely captured by the covariance matrix

$$\Sigma \stackrel{\text{def}}{=} (\mathrm{Cov}(Z_i, Z_j))_{\substack{1\le i\le n\\ 1\le j\le n}}.$$

This kind of dependency is easily treated by a simple linear transformation on the Y_i's. Indeed, there exists a unitary matrix $P = (P_{ij})_{\substack{1\le i\le n\\ 1\le j\le n}}$ such that $P\Sigma P^t$ is diagonal and takes the following form

$$P\Sigma P^t = \begin{pmatrix} {\sigma'_1}^2 & 0 & \cdots & & \\ 0 & {\sigma'_2}^2 & 0 & \cdots & \\ & & \cdots & & \\ & & \cdots & 0 & {\sigma'_n}^2 \end{pmatrix}$$

This implies that the vector $(Z'_i)_{1\le i\le j}$ where $Z'_i \stackrel{\text{def}}{=} \sum_{1\le j\le n} P_{ij} Z_j$ is a Gaussian vector formed by independent Gaussian variables of variance ${\sigma'_i}^2$. It would be straightforward to extend the approach followed here to the dependent case by performing the linear transformation associated to P. We would have to consider a slightly more general channel than the binary input additive white Gaussian noise channel, because we would have to calculate the corresponding linear transformations on the counters, namely

$$Y_i' \overset{\text{def}}{=} \sum_{1 \le j \le n} P_{ij} Y_j = \sum_{1 \le i \le n} P_{ij} (-1)^{\mathbf{k} \cdot \kappa_j} + Z_i'.$$

It should be added that for the 8-round DES, experiments have been handled in [GT09] to show that these covariances are negligible and that the Z_i's are essentially independent random variables. To simplify the discussion, from now on, we will assume that the Z_i's are independent random variables.

3.2. Maximum likelihood list decoding

The natural way of handling the analysis phase in linear cryptanalysis is to compute the likelihoods (or at least an equivalent statistic) of all the candidates and to choose the most likely key. In coding theory, this process is known as a *complete maximum likelihood decoding* algorithm.

Definition 4 *A* complete maximum likelihood decoding *algorithm is given a vector* $\mathbf{y}$ *for input and returns a codeword* $\mathbf{c}_{\text{ret}}$ *that maximizes the probability of being sent knowing the output* $\mathbf{y}$ *of the channel:*

$$\mathbf{c}_{\text{ret}} \overset{def}{=} \underset{\mathbf{c} \in \mathcal{C}}{\text{argmax}} \Pr\left[\mathbf{X} = \mathbf{c} | \mathbf{Y} = \mathbf{y}\right].$$

In our scope, the cryptanalyst is able to check the correctness of a codeword [2]. Thus, getting not one but the ℓ most likely codewords may be of interest. Although it makes less sense in coding theory to get many codewords at the output of a decoder, this kind of problem is well known as *list decoding.*

Definition 5 *A* list decoding *algorithm takes the received vector* $\mathbf{y}$ *for input and returns a list* $\mathcal{L} = \{\mathbf{c_1}, \mathbf{c_2}, \dots, \mathbf{c}_\ell\}$ *of the* ℓ *most likely codewords.*

$$\forall \mathbf{c_i} \in \mathcal{L}, \forall \mathbf{c} \in \mathcal{C} \backslash \mathcal{L},\ \Pr[\mathbf{X} = \mathbf{c_i} | \mathbf{Y} = \mathbf{y}] \ge \Pr[\mathbf{X} = \mathbf{c} | \mathbf{Y} = \mathbf{y}].$$

The following section presents two maximum likelihood list decoding algorithms that can be applied to the linear code $\mathcal{C}_{\kappa_1, \dots, \kappa_n}$ considered in a multiple linear cryptanalysis.

[2]Indeed, a codeword corresponds to a class of keys that can be used to encrypt a plaintext for which the ciphertext is known.

4. Finding bits of the key using a list decoding algorithm of a linear code

Regarding the model presented in the previous section, multiple linear cryptanalysis can be addressed as a maximum likelihood list decoding problem of the linear code generated by the key masks over a Gaussian channel. The point is that decoding algorithms may reduce significantly the cost of finding the most likely key. For instance, the method using ellipsoids [Dum00] works with complexity $O\left(2^{\frac{d(n-d)}{n}}\right)$ where d is the dimension of the vector space spanned by the κ_i's, if we bound and quantize the Y_i's. This is strictly better than the naive algorithm consisting in trying all the possible keys.

We present here two other algorithms that have been suggested in literature to decrease the complexity of the naive approach. The first one is an efficient global computation of all the likelihoods using a Walsh transform that is actually a complete soft decoding algorithm of first order Reed-Muller code over an erasure channel [FLT09a,FLT09b]. This algorithm is deterministic and provides the list of all codewords and their likelihood with a complexity of $O(d\,2^d)$ but requires to make an assumption of stochastic independence on the approximations used. The second one is a soft list decoding algorithm for random linear codes [Val00b][3]. Here, not all the codewords' likelihoods are computed: only a subset of potentially correct codewords are treated. This algorithm is thus a probabilistic algorithm that returns a list of the ℓ most likely codewords considered. This last algorithm fails when the correct codeword is not considered[4].

4.1. Decoding Reed-Muller codes on the erasure channel

It is straightforward to check (see for instance [GT09]) that the quantity $v(\mathbf{c})$ defined below induces the same ordering on the codewords $\mathbf{c}$'s as the probabilities $\Pr[\mathbf{X} = \mathbf{c} | \mathbf{Y} = \mathbf{y}]$ if the channel is **memoryless**.

$$v(\mathbf{c}) \overset{\text{def}}{=} \sum_{i=1}^{n} (-1)^{\mathbf{c}_i} \, \frac{Y_i}{\sigma_i}. \tag{2}$$

In order to compute this quantity, $O(n)$ sums and floating point divisions have to be performed thus the cost of individually computing the likelihood of each single codeword may be prohibitive (of order $O(n\,2^d)$).

The main tool used here is the Walsh transform of a real valued function g that maps $\mathbb{F}_2^d$ to $\mathbb{R}$. We denote by $\hat{g}$ this function that is defined as

$$\hat{g}(\mathbf{t}) \overset{\text{def}}{=} \sum_{\mathbf{a} \in \mathbb{F}_2^d} (-1)^{\mathbf{a}\cdot\mathbf{t}} \, g(\mathbf{a}).$$

[3]More details are given in Valembois' thesis [Val00a].

[4]If the correct codeword is considered but is not in the list of the ℓ most likely codewords, then any algorithm will fail. On the other hand, it is possible that the second algorithm succeeds where the first fails if, by chance, some codewords more likely than the correct one are not considered. If this second algorithm has a low failure probability, then this last event is unlikely to occur.

This function can be efficiently computed using the Fast Walsh Transform Algorithm. The complexity of this algorithm is of order $O(d\,2^d)$.

The link to the decoding of Reed-Muller codes is the following. The code generated by all the key masks of the subspace spanned by the n masks κ's is the order one Reed-Muller code of dimension d. The vector $\mathbf{Y}$ can be extended to a vector $\hat{\mathbf{Y}}$ of length 2^d by adding zeros corresponding to the key masks belonging to the aforementioned subspace that have no corresponding counters. This corresponds to receiving a vector that passes through a channel with erasures (the corresponding Y_i's are set to 0). This vector $\hat{\mathbf{Y}}$ is then a noisy vector corresponding to an initial codeword of the order 1 Reed-Muller code of dimension d for which an efficient complete soft decoding algorithm consists in performing the Fast Walsh Transform Algorithm.

The key masks κ_i span a vectorial subspace of $\mathbb{F}_2^k$ of dimension d. This one can, thus, be identified to $\mathbb{F}_2^d$ thanks to an isomorphism $\Psi : \mathbb{F}_2^d \to \mathcal{K} \subset \mathbb{F}_2^k$. Let us define

$$g(\mathbf{a}) \overset{\text{def}}{=} \begin{cases} Y_i/\sigma_i & \text{if } \exists i, \Psi(\mathbf{a}) = \kappa_i, \\ 0 & \text{else.} \end{cases}$$

Then, the Walsh transform $\hat{g}$ of g applied to a key class $\mathbf{k} \in \mathbb{F}_2^{\mathbf{d}}$ is

$$\hat{g}(\mathbf{k}) = \sum_{i=1}^{n} (-1)^{\kappa_i \cdot \mathbf{k}} \frac{Y_i}{\sigma_i}.$$

This is precisely the likelihood of a key class and thus the Fast Walsh Transform Algorithm performs the analysis phase of a multiple linear cryptanalysis in $O(d\,2^d)$ time. Notice that this complexity does not depend on n the number of approximations but on d the dimension of the subspace spanned by the key masks. Thus, adding approximations with a key mask that belongs to this subspace only increases the counters generation but not the complexity of computing likelihoods.

4.2. Stochastic resonance algorithm

Using Fast Walsh Transform Algorithm decreases the complexity of computing the likelihoods of the candidates. Nevertheless, when the number of candidates is too large, increasing the speed of exhaustive search will not be enough. In this case, the attacker has to skip candidates for the attack to have a reasonable time complexity.

The way of choosing which likelihoods will be computed is a soft decoding problem. Here the code $\mathcal{C}$ we consider is defined by

$$C = \{(\kappa_i \cdot \mathbf{k})_{1 \le i \le n}; \mathbf{k} \in \{0,1\}^k\}$$

This code has a priori no particular structure and thus, the problem is the one of decoding a random linear code that is known to be hard.

Nevertheless, there exists a probabilistic algorithm of relatively low exponential complexity for performing maximum likelihood decoding or list decoding with small list sizes. This algorithm is the Stochastic Resonance Decoding Algorithm that has been presented in [Val00b]. We denote by d the dimension of the code (that corresponds to the number of information bits recovered) and n its length (the number of approximations). We present this algorithm in what follows.

Basically the algorithm tries to look only for "rather" likely candidates. This is performed by considering a set of $d+h$ “rather reliable” positions. The algorithm is summarized by Algorithm 3.

Algorithm 3 Stochastic Resonance Algorithm [Val00a,Val00b].

Parameters:

- r a certain number of iterations
- ℓ a list size
- h a small parameter (typically smaller than 20)
- p (typically either 1 or 2)

Input: $\mathbf{Y}$ the received vector.
Output : a subset $\mathcal{L}$ contains the ℓ most likely codewords met by the algorithm

```
1: $\mathcal{L} \to \emptyset$.
2: for $1 \leq t \leq r$ do
3:     Step 1: A set $I$ of $d+h$ “rather reliable” positions is chosen
4:     Step 2: A vector $\tilde{y}_I$ consisting in the most likely values for $\kappa_i \cdot \mathbf{k}$ for $i$
       belonging to $I$ is computed:
5:     for $i \in I$ do
6:         if $Y_i > 0$ then $\tilde{y}_I(i) = 0$
           else $\tilde{y}_I(i) = 1$
7:     end for
8:     Step 3: A list $\mathcal{C}(2p, I, \tilde{y}_I)$ of codewords (i.e. elements of $C$) is computed
       such that any codeword $\mathbf{c}$ in it disagrees with $\tilde{y}_I$ on at most $2p$ positions of
       $I$, that is $|\{i \in I; \mathbf{c}_i \neq \tilde{y}_I(i)\}| \leq 2p$
9:     Step 4:
10:    for all $\mathbf{c} \in \mathcal{C}(2p, I, \tilde{y}_I)$ do
11:        compute its reliability $v(\mathbf{c})$
12:        if $|\mathcal{L}| < \ell$ then $\mathcal{L} = \mathcal{L} \cup \{\mathbf{c}\}$
           else if $v(\mathbf{c})$ larger than the smallest reliability of the elements of $\mathcal{L}$, replace
           the element in $\mathcal{L}$ of smallest reliability by $\mathbf{c}$
13:    end for
14: end for
15: return $\mathcal{L}$
```

The first step for obtaining sets of rather reliable positions (but which differ substantially from each other) uses the recipe given by Algorithm 4.

Let us now explain how the third step works. The idea is that the most likely codeword has decent chances to agree on most positions of I with $\tilde{y}_I$ and therefore to try all $\sum_{j=1}^{2p} \binom{d+h}{j}$ possibilities of codewords if we check all possibilities of up to $2p$ disagreements on I. Note that not all these possibilities might be valid

Algorithm 4 Stochastic way of producing "good" subsets I [Val00a,Val00b].

```
Parameters: $h$ a small parameter (typically smaller than 20)
Input: $\mathbf{Y}$ the received vector.
Output : a subset $I \subset \{1, \dots, n\}$ such that $|I| = d+h$
1: for $1 \leq i \leq n$ do
2:     $Y_i' \longleftarrow Y_i + N_i$ with $N_i$ a Gaussian variable with expected value 0 and
       variance $\frac{\sigma_i^2}{4}$
3: end for
4: Sort the $Y_i'$'s according to their reliability $|Y_i'|$
5: $I \longleftarrow d + h$ most reliable positions
6: return $I$
```

codewords. Indeed, if we let C_I be the projection π_I of C onto the coordinates that are in I which is given by the mapping

$$\begin{aligned}\pi_I : \mathbb{F}_2^n &\to \mathbb{F}_2^{|I|} \\ \mathbf{x} = (x_j)_{1 \leq i \leq n} &\mapsto \mathbf{x}_I \overset{\text{def}}{=} (x_i)_{i \in I}\end{aligned}$$

then we would like to check only the elements which disagree with $\tilde{y}_I$ in at most $2p$ positions and which are in C_I. There is a way to check a good fraction of these possibilities with a complexity only of order $\sum_{j=1}^{p} \binom{\lfloor \frac{d+h}{2} \rfloor}{j}$ if h is such that

$$\sum_{j=1}^{p} \binom{\lfloor \frac{d+h}{2} \rfloor}{j} = O(2^h), \tag{3}$$

by splitting I into two subsets I_1 and I_2 of almost equal size and to look only at codewords of C_I which disagree with $\tilde{y}_I$ in at most p positions of I_1 and p positions in I_2. The point is the following. We can find with standard Gaussian elimination a *parity-check matrix* for C_I, namely a matrix M whose kernel is equal to C_I:

$$C_I = \{\mathbf{x} \in \mathbb{F}_2^{|I|}; M\mathbf{x} = 0\}.$$

In other words, we want to find all possible $e_1, e_2 \in \mathbb{F}_2^{|I|}$ both of weight $\leq p$, with the support of e_i being contained in I_i, such that

$$M(\tilde{y}_I \oplus e_1 \oplus e_2) = 0.$$

We may perform this operation by using a hash table as detailed in Algorithm 5. It remains to explain how we perform Instruction 10 in Algorithm 5, that is how we extend a codeword in C_I into a codeword of C. This is performed by computing by standard Gaussian elimination a matrix G of size $|I| \times n$ such that the right multiplication of an element $\mathbf{x}_I$ of C_I yields the element of C whose projection by π_I is $\mathbf{x}_I$. Such a matrix always exists if the dimension of C_I is d. This happens with probability very close to 1 for h satisfying Bound (3).

Algorithm 5 An algorithm to generate $\mathcal{C}(2p, I, \mathbf{y})$.

Parameters: $2p$ a bound on the number of positions in I on which the input and the elements of the output differ.
Input:

- a subset $I \subset \{1, 2, \ldots, n\}$ with $|I| = d + h$
- a vector $\mathbf{y} = (y_i)_{i \in I} \in \mathbb{F}_2^{d+h}$

Output : a list of elements $\mathbf{c}$ of C such that $|\{i \in I; \mathbf{c}_i \neq y_i\}| \leq 2p$

```
1: $\mathcal{C} \longleftarrow \emptyset$
2: Initialise an empty hashtable $H[\cdot]$ of length $2^h$
3: Find a parity-check matrix $M$ for $C_I$
4: Randomly split $I$ in two sets $I_1$ and $I_2$ such that $||I_1| - |I_2|| \leq 1$
5: for all possible $\mathbf{e_1} \in \mathbb{F}_\mathbf{2}^{\mathbf{d+h}}$ of support in $I_1$ and weight $\leq p$ do
6:     $H[M\mathbf{e}_1] \leftarrow H[M\mathbf{e}_1] \cup \mathbf{e}_1$
7: end for
8: for all possible $\mathbf{e_2} \in \mathbb{F}_\mathbf{2}^{\mathbf{d+y}}$ of support in $I_2$ and weight $\leq p$ do
9:     for all $\mathbf{e}_1$ in $H[M\mathbf{e}_2 \oplus M\mathbf{y}]$ do
10:        $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{extend}(\mathbf{e}_1 \oplus \mathbf{e}_2 \oplus \mathbf{y})\}$;
11:    end for
12: end for
13: return $\mathcal{C}$
```

4.3. Comparison of the two algorithms

Let us have a small discussion on the choice of the algorithm to use when we want to perform multiple linear cryptanalysis. If the dimension d is small enough, the first algorithm of Subsection 4.1 is clearly the one to use. However, if d is too large for $d\,2^d$ to be negligible regarding the final exhaustive search of the attack (for instance if the κ_i's span all the key space), then, the first algorithm has prohibitive complexity and Algorithm 3 should be preferred. There is no known complexity analysis known for this algorithm, but it is expected to run successfully to find the most likely candidate with an exponential complexity which is significantly smaller than the ellipsoid method of [Dum00] which runs with complexity $O\left(2^{\frac{d(n-d)}{n}}\right)$.

5. Estimating or Bounding the data complexity using entropy

In a statistical cryptanalysis, the rank of the correct key among the list of all existing key classes is an essential factor, for instance the time complexity depends heavily on it through the list size ℓ of the key candidates which are kept. Two notions can be used to quantify the power of an attack.

- The *advantage* **adv**: in [Sel08], the advantage of an attack for a success probability P_S is defined as $\log_2\left(\frac{|\mathcal{Z}|}{\ell}\right)$ where $|\mathcal{Z}|$ is the total number of key classes considered by the attack, whereas ℓ is such that keeping ℓ candidates among $|\mathcal{Z}|$ will lead to a success probability P_S. Since for reasonable P_S, the advantage of an attack does not vary so much, an alternative definition can be used which does not depend on P_S (see [HCN09] for instance). We will use this alternative definition, that is we take ℓ in the formula above such that $P_S = 0.5$.

- The *gain* : this quantity is defined in [BCQ04] and is derived from the expected value of the rank R of the correct key by the formula $\log_2 \frac{|\mathcal{Z}|}{2\mathbb{E}(R)-1}$.

These quantities seem to be quite similar. Indeed, we may write ℓ as the list size such that

$$P_S \overset{\text{def}}{=} \Pr(\mathrm{R} \leq \ell) = \frac{1}{2}.$$

In other words, ℓ is the median of R, which we denote by $\mathbf{median}(R)$, and we may expect that the median is roughly equal to $\mathbb{E}(R)$ (i.e. the expectation of R). Therefore, we may think that

$$\mathbf{adv} = \log_2 \frac{|\mathcal{Z}|}{\mathbf{median}(R)} \approx \log_2 \frac{|\mathcal{Z}|}{\mathbb{E}(R)} \approx \log_2 \frac{2|\mathcal{Z}|}{2\mathbb{E}(R)-1} \approx \mathbf{gain} + 1.$$

Unfortunately, this intuition turns out to be wrong in many cases. The reason for this is that R is rather concentrated and the median of it is not far away from its typical value. However the expectation of R is in general far away from the median. This is related to the fact that R is in general a very large quantity which is in many cases of interest exponential in the key size k. Very often, the expectation turns out to be dominated in such cases by *rare events*: in certain rather unlikely cases, R is quite larger than the median of R and this is taken heavily into account into the expectation and leads to an expectation $\mathbb{E}(R)$ which is significantly larger than $\mathbf{median}(R)$.

We present in Proposition 1 which follows an example where this phenomenon occurs. Basically for the multilinear cryptanalysis attack analyzed there, the advantage is equal to the whole key size d (meaning that with success probability of $\frac{1}{2}$ the best ranked key is the right one, that is $\ell = 1$) for a number of plaintext/ciphertexts pairs of order

$$N \approx \frac{d \ln(2)}{2 \sum_{i=1}^{n} \epsilon_i^2},$$

where the ϵ_i's are the biases of the n linear approximations used for the attack. In other words, for N of this order, we have $\mathbf{median}(R) \approx 1$. However, the calculations of [BCQ04] show that $\mathbb{E}(R)$ is exponential in d for such values of N. It becomes of order $O(1)$ only for values of N which are twice as large.

On the other hand, the advantage can be quite hard to compute. This can be illustrated, for instance, by the case of multidimensional linear cryptanalysis of type 1 [HCN09c] where the computation of the advantage requires to compute an order statistic of 2^d dependent Gaussian variables with different negative expected values and same variance. In that case the only way of deriving a formula for the advantage is to make some simplifying hypotheses. The formula for the advantage in [HCN09c] is actually derived assuming that all these variables are independent and identically distributed. Moreover, the common distribution taken is the Gaussian distribution with expected value 0 what is clearly a pessimistic assumption.

A way of overcoming the problem of the bad behavior of the expectation of R is to take the expectation of other quantities related to R. For instance, it is shown in [GT09], that considering the entropy of the key is a much better statistic than the expectation of the rank R. This entropy measures the number of random bits remaining in the key knowing certain statistics. In a certain sense, it is related to the expectation of the logarithm $\log_2(R)$. The logarithm of R varies much less than R and this is why the typical size of $\log_2(R)$ coincides quite well with the expectation. Calculating the expectation of $\log_2(R)$ is probably a very tough task. However, it is relatively easy to estimate the entropy by using a well known information theoretical inequality, namely Theorem 1. This theorem gives a lower bound on the entropy which is very easy to compute even in rather complicated cases and is surprisingly sharp in general.

By using this theorem we obtain lower bounds for the key entropy for various kinds of linear cryptanalyses. Our formula is compared to the data complexity given in [BCQ04] for multiple linear cryptanalysis MK1 and it turns out that our resulting formula gives an estimate smaller by a factor of 2 than the one given in [BCQ04]. Then, its application to simple linear cryptanalysis is considered. The result obtained there can be put in perspective with a remark of Pascal Junod in [Jun01]: *"We observe that Theorem 1 seems to give a pessimistic rank expected value. It is difficult to explain this fact because of the small statistical sample size".*

To conclude, this approach is applied to the multidimensional cryptanalysis of SERPENT presented in [HCN09]. In this case also, the entropy approach is closer to the experimental results than the theoretical estimates given by the attack designers.

5.1. Estimating the key entropy

The *entropy* of a random variable quantifies its uncertainty. It will give a very handy tool to quantify the information gained analysing plaintext/ciphertext pairs.

Definition 6 *The (binary)* entropy $\mathcal{H}(X)$ *of a random variable* X *is given by the expression:*

$$\mathcal{H}(X) \overset{def}{=} -\sum_{x} \Pr[\mathrm{X = x}] \log_2 \Pr[\mathrm{X = x}] \textit{ (for discrete } X\textit{)}$$

$$\overset{def}{=} -\int f(x) \log_2 f(x) dx \textit{ (for continuous } X \textit{ of density } f\textit{).}$$

Definition 7 *The conditional (binary)* entropy $\mathcal{H}(X|Y)$ *of a random variable* X *under the knowledge of another random variable* Y *is given by the expression:*

$$\mathcal{H}(X|Y) \overset{def}{=} \sum_{x,y} \Pr[\mathrm{Y = y}]\mathrm{H(X|Y = y)} \textit{ (for discrete } Y\textit{)}$$

$$\overset{def}{=} \int f(y) H(X|Y = y) dy \textit{ (for continuous } Y\textit{).}$$

where

$$\mathcal{H}(X|Y=y) \overset{def}{=} -\sum_x \Pr[\mathrm{X}=\mathrm{x}|\mathrm{Y}=\mathrm{y}] \log_2 \Pr[\mathrm{X}=\mathrm{x}|\mathrm{Y}=\mathrm{y}] \text{ (for discrete } X\text{)}$$

$$\overset{def}{=} -\int f(x|y)\log_2 f(x|y)dx \text{ (for continuous } X\text{)}.$$

Conditional entropy can be used to measure the amount of randomness (i.e. the amount of random bits) left in a key given a certain amount of statistics derived from plaintext/ciphertext pairs. More precisely, if we denote by $\mathbf{K}$ denote the random variable $\mathbf{K} = (\kappa_i \cdot \mathbf{k})_{1\le i\le n}$ and by $\mathbf{Y}$ the random variable that contains the information extracted from the available samples, then $\mathcal{H}(\mathbf{K}|\mathbf{Y})$ quantifies the uncertainty the attacker has on $\mathbf{K}$ knowing the plaintext/ciphertext pairs. For instance, if $\mathcal{H}(\mathbf{K}|\mathbf{Y}) = 0$, the attacker has a complete knowledge of $\mathbf{K}$ and thus the correct candidate is top ranked with probability 1. More generally, taking $\ell = 2^{\mathcal{H}(\mathbf{K}|\mathbf{Y})}$ should be enough to reach a good success probability.

The *mutual information* $\mathcal{I}(X;Y)$ between X and Y is a related quantity which quantifies the information on X brought by the knowledge of Y. It is defined by

$$\mathcal{I}(X;Y) \overset{\text{def}}{=} \mathcal{H}(X) - \mathcal{H}(X|Y). \tag{4}$$

It is straightforward to check [CT91] that this quantity is symmetric and that

$$\mathcal{I}(X;Y) = \mathcal{I}(Y;X) = \mathcal{H}(Y) - \mathcal{H}(Y|X).$$

5.2. The main tool

The main tool used in [GT09] to estimate the conditional expectation of the key is the following theorem which relates the whole entropy to quantities which are easier to compute.

Theorem 1 (Lemma 1 of [GT09]) *Let* $\mathbf{Y}$ *and* $\mathbf{K}$ *be two vectors of* n *random variables such that*

$$f(\mathbf{Y}|\mathbf{K}) = \prod_{i=1}^{n} f(Y_i|K_i), \tag{5}$$

where $f(\mathbf{Y}|\mathbf{K})$ *denotes either the conditional probability of* $\mathbf{Y}$ *given* $\mathbf{K}$ *when* $\mathbf{Y}$ *is discrete or the conditional density when* $\mathbf{Y}$ *is continuous. Then,*

$$\mathcal{H}(\mathbf{K}|\mathbf{Y}) \ge \mathcal{H}(\mathbf{K}) - \sum_{i=1}^{n} \mathcal{I}(K_i;Y_i). \tag{6}$$

Since keeping a list of $2^{\mathcal{H}(\mathbf{K}|\mathbf{Y})}$ candidates guarantees a success probability close to 1 and since $\mathcal{I}(K_i;Y_i)$ is an increasing function of N, this lower bound on the entropy is a lower-bound on the number N of plaintext/ciphertext pairs required for a successful cryptanalysis.

This theorem will often be applied together with the following observation (which is straightforward consequence of the definition of the conditional entropy)

lemma 1 *If g is a one to one mapping, then $\mathcal{H}(\mathbf{K}|\mathbf{Y}) = \mathcal{H}(g(\mathbf{K})|\mathbf{Y})$.*

The point of this lemma is that it even if the vectors $\mathbf{K}$ and $\mathbf{Y}$ do not satisfy the conditional independence relation (5), we might be able to choose a one-to-one mapping g such that

$$f(\mathbf{Y}|g(\mathbf{K})) = \prod_{i=1}^{n} f(Y_i|K_i')$$

where K_i' is the i-th coordinate of $g(\mathbf{K})$.

5.3. Application to various linear cryptanalyses

We give here several applications of Theorem 1 to various cryptanalyses that can be found in [GT09,Gér10]. It should be mentioned that Theorem 1 has a much broader range of applications than the four examples which are given here (it could be applied to many other scenarios in statistical cryptanalysis for instance).

Attack 1. This attack corresponds to the first attack proposed in [Mat94] and to algorithm MK1 of [BCQ04]. For each linear approximation, a counter T_i containing the number of couples $(\mathbf{p}^j, \mathbf{c}^j)$ such that $\mathbf{p}^j \cdot \pi \oplus \mathbf{c}^j \cdot \gamma = 0$ is computed and we want to recover the key $\mathbf{K}$ from these counters. We are exactly in the setting explained in Subsection 3.1. We use Theorem 1 with

$$Y_i \overset{\text{def}}{=} (-1)^{\mathbf{k}\cdot\kappa_i} + Z_i$$
$$K_i \overset{\text{def}}{=} (-1)^{\mathbf{k}\cdot\kappa_i}$$

and obtain (for more details, see [GT09])

$$\mathcal{H}(\mathbf{K}|\mathbf{Y}) \geq d - \sum_{i=1}^{n} \mathbf{Cap}(\sigma_i^2), \tag{7}$$

where d is the dimension of the subspace spanned by the key masks κ_i's and $\mathbf{Cap}(\sigma_i^2)$ stands for the capacity of a BAWGN channel of variance σ_i^2. It is given by the following expression that can be found in [RU08] for instance

$$\mathbf{Cap}(\sigma^2) = 1 - \frac{\sigma}{\sqrt{8\pi}} \int_{-\infty}^{\infty} e^{-\frac{(\sigma^2 t - 2)^2}{8\sigma^2}} \log_2(1 + e^{-t})\, dt;$$

It can be expanded in the following series

$$\mathbf{Cap}(\sigma^2) = \frac{1}{\ln(2)} \left(\frac{1}{2\sigma^2} - \frac{1}{4\sigma^4} + \frac{1}{6\sigma^6} + O\left(\sigma^{-8}\right) \right).$$

Attack 2 and 3. The second kind of linear cryptanalyses are distinguishing attacks. In such attacks, the attacker uses a linear approximation over $r - r'$ rounds of the cipher where r is the targeted number of rounds (i.e. samples are encrypted using r rounds of the cipher) and r' is a small number of rounds chosen by the cryptanalyst. This kind of attacks have been proposed by Matsui as Algorithm 2 and have been extended to multiple linear cryptanalysis under the name Algorithm MK2 in [BCQ04]. In these attacks, the attacker recovers both the outer key $\mathbf{k}^{\mathrm{O}}$, that is key bits involved in the r' last rounds and the inner key $\mathbf{k}^{\mathrm{I}}$ that is the one recovered in Attack 1. Attack 2 is a distinguishing attack where only $\mathbf{k}^{\mathrm{O}}$ is recovered and Attack 3 corresponds to the cryptanalyses in [Mat94,BCQ04] where both $\mathbf{k}^{\mathrm{O}}$ and $\mathbf{k}^{\mathrm{I}}$ are considered. It is important to differentiate these two attacks since Attack 3 might not be applicable to ciphers with complex key schedules where the key masks do not make sense while Attack 2 does not consider these masks.

In both case, for each of the n approximations, ciphertexts are partially decrypted with all possible values for the bits of the outer key involved in the process. Notice that the bits involved may not be the same from an approximation to another. We denote by $\mathcal{Z}_i$ the set of possible values for the bits of the outer key involved in the partial decryption corresponding to the i-th approximation and by $\mathbf{k}_i^{\mathrm{O}}$ the part of the outer key involved in it. Then, a counter can be computed for each of these possible values of $\mathbf{k}^{\mathrm{O}}$. For the i-th approximation, we denote by Y_i^z the statistic obtained using the value $\mathbf{k}_i^{\mathrm{O}} = z$.

In the case of Attack 2, we define

$$\begin{aligned}
\mathbf{K} &\stackrel{\text{def}}{=} \mathbf{k}^{\mathrm{O}} \\
g(\mathbf{K}) &\stackrel{\text{def}}{=} (\mathbf{k}_{i,z}^{\mathrm{O}})_{\substack{1\leq i\leq n \\ z\in\mathcal{Z}_i}} \\
\text{with } \mathbf{k}_{i,z}^{\mathrm{O}} &\stackrel{\text{def}}{=} \mathbf{k}_i^{\mathrm{O}} \\
\mathbf{Y} &\stackrel{\text{def}}{=} (Y_i^z)_{\substack{1\leq i\leq n \\ z\in\mathcal{Z}_i}}
\end{aligned}$$

We apply Theorem 1 to $g(\mathbf{K})$. If we denote by $2^{k^{\mathrm{O}}}$ the number of possible values for $\mathbf{K}$, we obtain (for more details see [GT09])

$$\mathcal{H}(\mathbf{K}|\mathbf{Y}) \geq k^{\mathrm{O}} - \sum_{i=1}^{n} \sum_{z\in\mathcal{Z}_i} \mathcal{I}(\mathbf{k}_i^{\mathrm{O}}; Y_i^z).$$

If we denote by $\varphi_i^\alpha(t) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left[-\frac{(t-\alpha)^2}{2\sigma_i^2}\right]$, then, the formula for $\mathcal{I}(\mathbf{k}_i^{\mathrm{O}}; Y_i^z)$ is, for Attack 2:

$$\mathcal{I}(\mathbf{k}_i^{\mathrm{O}}; Y_i^z) = \int_0^\infty \frac{r_i(t)}{|\mathcal{Z}_i|} \log\left(\frac{r_i(t)}{s_i(t)}\right) dt + \int_0^\infty (1 - |\mathcal{Z}_i|^{-1}) w_i(t) \log\left(\frac{w_i(t)}{s_i(t)}\right) dt,$$

with $r_i(t) = \varphi_i^1(t) + \varphi_i^{-1}(t)$, $w_i(t) = 2\varphi_i^0(t)$ and $s_i(t) \stackrel{\text{def}}{=} |\mathcal{Z}_i|^{-1} r_i(t) + (1 - |\mathcal{Z}_i|^{-1}) w_i(t)$.

In the case of Attack 3, we obtain in a similar way (see [GT09])

$$\mathcal{H}(\mathbf{K}|\mathbf{Y}) \geq k^{\mathrm{O+I}} - \sum_{i=1}^{n} \sum_{z \in \mathcal{Z}_i} I_{i,z}.$$

where $2^{k^{\mathrm{O+I}}}$ is the total number of possible pairs $(\mathbf{k}^{\mathrm{O}}, \mathbf{k}^{\mathrm{I}})$ and

$$I_{i,z} = \int_{-\infty}^{\infty} \frac{\varphi_i^1(t)}{|\mathcal{Z}_i|} \log\left(\frac{\varphi_i^1(t)}{\psi_i(t)}\right) dt + \int_{-\infty}^{\infty} (1 - |\mathcal{Z}_i|^{-1}) \varphi_i^0(t) \log\left(\frac{\varphi_i^0(t)}{\psi_i(t)}\right) dt,$$

with $\psi_i(t) \overset{\mathrm{def}}{=} (1 - |\mathcal{Z}_i|^{-1})\, \varphi_i^0(t) + |\mathcal{Z}_i|^{-1}\, \dfrac{\varphi_i^{-1}(t) + \varphi_i^1(t)}{2}$.

Multidimensional linear cryptanalysis. The last application of Theorem 1 we present is the case of multidimensional cryptanalysis [HCN09b,HCN08]. We will later compare the obtained results to experimental results provided in [HCN09].

Here, d base approximations are considered. Hence, for each pair $(\mathbf{p}^j, \mathbf{c}^j)$, we can derive d counters

$$T_i^j \overset{\mathrm{def}}{=} \begin{cases} 1 & \text{if } \mathbf{p}^j \cdot \pi_i \oplus \mathbf{c}^j \cdot \gamma_i = 0, \\ 0 & \text{otherwise.} \end{cases}$$

Both extensions of Algorithm 1 and 2 are based on the study of the distribution of $T^j \overset{\mathrm{def}}{=} (T_1^j, \ldots, T_d^j) \in \mathbb{F}_2^d$. This distribution depends on the value of the inner key $\mathbf{k}^{\mathrm{I}}$ and thus is denoted by $p_{\mathbf{k}^{\mathrm{I}}}$. If we bring in $\mathbf{Y} = (T^1, \ldots, T^n)$, $\mathbf{K} = \mathbf{k}^{\mathrm{I}}$ we can apply Theorem 1 to the couple $(g(\mathbf{K}), \mathbf{Y})$ with $g(\mathbf{K}) \overset{\mathrm{def}}{=} (\underbrace{\mathbf{k}^{\mathrm{I}}, \ldots, \mathbf{k}^{\mathrm{I}}}_{N \text{ times}})$, what leads to

$$\mathcal{H}(\mathbf{K}|\mathbf{Y}) \geq d - \sum_{j=1}^{N} \mathcal{I}(\mathbf{k}^{\mathrm{I}}; T^j). \tag{8}$$

Using the properties of the distribution of T^j, it is straightforward to check the following expression which works for both multidimensional versions of Attack 1 and 3 (the whole derivation can be found in [Gér10]).

$$\mathcal{I}(\mathbf{k}^{\mathrm{I}}; T^j) = d - \mathcal{H}(T_j | \mathbf{k}^{\mathrm{I}} = 0). \tag{9}$$

Notice that this is the entropy of an uniformly distributed random variable on $\mathbf{F}_2^m$ minus the entropy of the distribution induced by the correlations. The conditional distribution of T_j given that $\mathbf{k}^{\mathrm{I}}$ is equal to 0 is straightforward to obtain and therefore we have an efficient way of lower bounding $\mathcal{H}(\mathbf{K}|\mathbf{Y})$.

5.4. Accuracy and relevance of the bound

The entropy approach is very general and really captures the good behavior of the rank of the good key. While for some special cases such as Matsui's algorithm

2, some significant work can lead to a very precise formula for the distribution of this rank ([Jun01]), there are cases of multilinear or multidimensional linear cryptanalysis where it is not known how to compute the advantage of the attack. The approach of [GT09] gives in this case a simple lower bound on the conditional entropy which is tight enough to give a good idea of the power of a cryptanalysis.

We propose here to emphasise the relevance and the accuracy of this approach for some linear cryptanalyses.

Entropy and list size. The conditional entropy $\mathcal{H}(\mathbf{K}|\mathbf{Y})$ measures the number of unknown key bits knowing the counters derived from the available samples. Thus, it is natural to suggest $2^{\mathcal{H}(\mathbf{K}|\mathbf{Y})}$ for the list size of candidates. It should be noted that in the case of the multilinear Attack 1, when our bound becomes negative a theoretical work based on typical joint sequences (Theorem 1 and Lemma 2 in [GT09]) proves that the probability that the right key is the best candidate is quite close to 1.

The accuracy of the bound (7) given for Attack 1 together with the fact that taking $\ell = 2^{\mathcal{H}(\mathbf{K}|\mathbf{Y})}$ induces a good success probability, was verified experimentally in [GT09]. For instance, an experimental cryptanalysis of the full DES was performed 19 times using 2^{39} pairs and 32968 approximations. In this setting, the bound (7) suggests to keep a list of size 2^{40}. The rank of the correct key in the 19 experiments are listed below.

$$2^{31.34}, 2^{33.39}, 2^{34.65}, 2^{35.24}, 2^{36.56}, 2^{37.32}, 2^{37.72}, 2^{37.99}, 2^{38.11}, 2^{38.52}, 2^{38.97},$$

$$2^{39.04}, 2^{39.19}, 2^{39.27}, 2^{39.53}, 2^{39.85}, 2^{40.28}, 2^{40.82}, 2^{40.88}.$$

We can see that in 3 out of 19 experiments, the attack fails that is a success probability of 0.84.

Attack 1. The accuracy of the bound on entropy is experimentally tested for Attack 1 in [GT09]. A toy cryptanalysis of 8-round DES using 76 linear approximations and recovering 13 bits of the key was performed. The curves in Figure 1 corresponds to the empiric entropy and the bound obtained on it as a function of the number of samples N. We can see that the bound is tight for small values of N (that are actually the values considered in cryptanalysis) while for bigger N it becomes slightly less precise. Nevertheless, the previous paragraph has shown that when the bound approaches zero, the probability of the correct key to be top ranked is close to 1 and thus this loss of precision is not as important as it could be thought at first sight.

Theorem 2 from [BCQ04] (Attack 1). The second thing we would like to emphasize is a comparison between the formulas obtained from both papers [BCQ04] and [GT09] in the case of top-ranking for Algorithm MK1 (Attack 1):

Proposition 1 *Proposition 3.1 in [GT09].*
Suppose that N is in a range where $\forall i, N\epsilon_i^2 = o(1)$. We consider the data complexity required to achieve top ranking on a d-bit key for Algorithm 1. Using (7) gives the following estimate

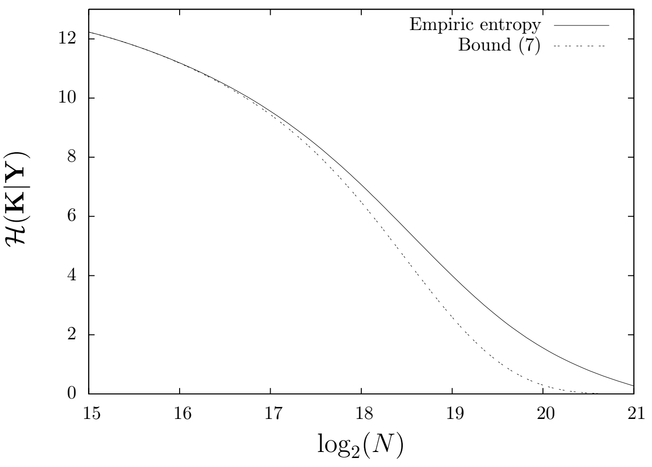

Figure 1. Theoretical and empirical entropies for a type 1 attack on 8-round DES [GT09].

$$N \approx \frac{d\ln(2)}{2\sum_{i=1}^{n}\epsilon_i^2}\left(1+o(1)\right). \tag{10}$$

The estimate obtained using Theorem 2 in [BCQ04] is

$$N \approx \frac{d\ln(2)}{\sum_{i=1}^{n}\epsilon_i^2}\left(1+o(1)\right). \tag{11}$$

The approximation given by (10) is sharp as shown in [GT09]. Notice the factor 2 between the two estimates. This is due to the fact that Theorem 2 in [BCQ04] is derived by estimating the gain of the attack that is, itself, derived from the expected rank of the correct key among the candidates. This is an illustration of the unfortunate behavior of the expected value of the rank in statistical cryptanalysis which was mentioned at the beginning of this section.

Matsui's Algorithm 2 [Mat94,Jun01] (Attack 2). Applying the formula for Attack 2 to Matsui's Algorithm 2, one obtains that for a time complexity of 2^{43}, 2^{41} pairs are enough. This is also what has been experimentally observed by Junod while his Theorem 1 in [Jun01] predicted a complexity of 2^{43}.

The theoretical framework provided by Junod about the distribution of the rank of the correct key class is really tight as shown by the experiments. For different values of ψ, theoretical and empirical probabilities of the rank to be less than ψ are really close. Nevertheless, when considering the expected rank, the relatively small number of experiments (21) induces a gap between expected and observed values. Since the distribution of ranks is concentrated, it is very unlikely that one in 21 experiments leads to a rank value not in the typical set of ranks. Hence, the mean of experimental ranks is close to the median and far from the expected value. The work of Junod [Jun01] is a good perfect illustration of this statement.

Multidimensional cryptanalysis [HCN09]. An illustration of the fact that the advantage of an attack may not be easy to compute is multidimensional cryptanalysis. The theoretical results obtained in [HCN09] differ significantly from the experimental values which are observed. We have presented a bound on the conditional entropy $\mathcal{H}(\mathbf{K}|\mathbf{Y})$ earlier in the section. Here, we propose to compare the results obtained using this bound to both theoretical and experimental results that can be found in [HCN09].

Keeping $2^{\mathcal{H}(\mathbf{K}|\mathbf{Y})}$ candidates is expected to give a cryptanalysis with a good probability of success. Thus, $\mathcal{I}(\mathbf{K};\mathbf{Y})$ that equals $d - \mathcal{H}(\mathbf{K}|\mathbf{Y})$ should be a good estimate of the advantage of an attack. We have plotted in Figure 2 both the empirical and the theoretical advantage given in [HCN09] together with the expression for $\mathcal{I}(\mathbf{K};\mathbf{Y})$ obtained using (8) and (9). The values taken in [HCN09] are the ones obtained by using the LLR method. The reason is that the information theoretical bound has been derived without taking care of the way the information is processed. Thus, it is a bound on the information obtained by an optimal cryptanalysis and has to be compared with the `LLR` method. The results can be seen in Figure 2. The bound on the mutual information $\mathcal{I}(\mathbf{Y};\mathbf{K})$ seems

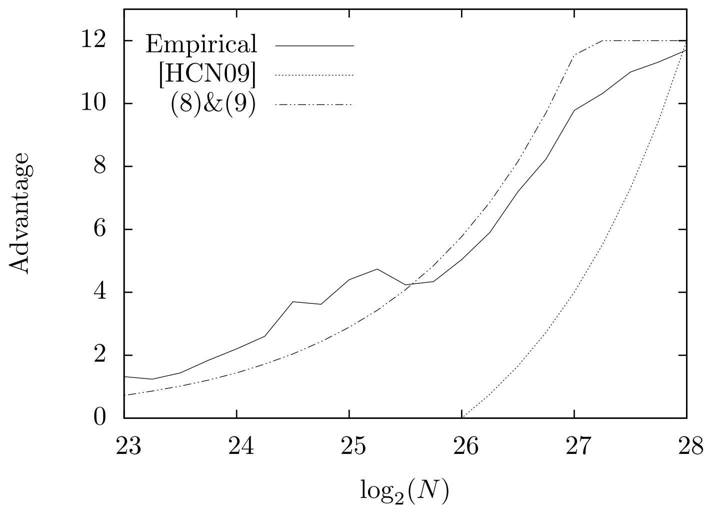

Figure 2. Theoretical and empirical advantages for multidimensional cryptanalysis of SERPENT with $m = 4$ [HCN09].

to be more suitable than the estimate for the advantage provided in [HCN09]. Notice that the mutual information is, for small N, smaller than the empirical advantage. This might be explained by the fact that the experimental curve is obviously very noisy (it is not even decreasing in N) which seems to be due to the fact that the number of experiments is not large enough to give an accurate experimental picture.

References

[CT91] Thomas M. Cover and Joy A. Thomas. *Information theory.* Wiley series in communications. Wiley, 1991.

[Mat94] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In Tor Helleseth, editor, *Advances in Cryptology - EUROCRYPT 1993*, volume 765 of *LNCS*, pages 386–397. Springer, 1994.

[Dum99] Ilya Dumer. Ellipsoidal lists and maximum likelihood decoding. *IEEE Transactions on Information Theory*, Vol. 46, No. 2, pages 649-656, March 2000.

[Val00a] Antoine Valembois. *Détection, Reconnaissance et Décodage des Codes Linéaires Binaires.* PhD thesis, Université de Limoges, 2000.

[Val00b] Antoine Valembois. Fast soft-decision decoding of linear codes, stochastic resonance in algorithms. In *IEEE International Symposium on Information Theory - ISIT 2000*, page 91, 2000.

[Jun01] Pascal Junod. On the complexity of Matsui's attack. In Serge Vaudenay and Amr M. Youssef, editors, *Selected Areas in Cryptography - SAC 2001*, volume 2259 of *LNCS*, pages 199–211. Springer, 2001.

[HKL03] Tor Helleseth, Torleiv Kløve, and Vladimir I. Levenshtein. Bounds on the error-correcting capacity of codes beyond half the minimum distance. In Daniel Augot, Pascale Charpin, and Grigorii Kabatiansky, editors, *Workshop on Coding and Cryptography - WCC 2003*, pages 243–251, 2003.

[BCQ04] Alex Biryukov, Christophe De Cannière, and Michaël Quisquater. On multiple linear approximations. In Yvo Desmedt, editor, *Advances in Cryptology - CRYPTO 2004*, volume 3152 of *LNCS*, pages 1–22. Springer, 2004.

[Mur06] Sean Murphy. The independence of linear approximations in symmetric cryptology. IEEE Transactions on Information Theory, Vol 52, pages 5510–5518, 2006.

[Sel08] Ali Aydin Selçuk. On probability of success in linear and differential cryptanalysis. *Journal of Cryptology*, 21(1):131–147, 2008.

[RU08] T. Richardson and R. Urbanke. *Modern Coding Theory.* Cambridge University Press, 2008.

[FLT09a] Rafaël Fourquet, Pierre Loidreau, and Cédric Tavernier. Finding good linear approximations of bock ciphers and its application to cryptanalysis of reduced round DES. In Alexander Kholosha, Eirik Rosnes, and Matthew Parker, editors, *Workshop on Coding and Cryptography - WCC 2009*, pages 501–515, 2009.

[FLT09b] Rafaël Fourquet, Pierre Loidreau, and Cédric Tavernier. Finding good linear approximations of bock ciphers and its application to cryptanalysis of reduced round DES. EUROCRYPT 2009 POSTERSESSION, 2009.

[RT09] Thomas Roche and Cédric Tavernier. Side-channel attacks based on linear approximations. In *International Alternative Workshop on Aggressive Computing and Security - iAWACS 2009.* ISBN Presses Techniques ESIEA, 2009.

[GT09] Benoît Gérard and Jean-Pierre Tillich. On linear cryptanalysis with many linear approximations. In Matthew G. Parker, editor, *IMA International Conference, Cryptography and Coding - IMACC 2009*, volume 5921 of *LNCS*, pages 112–132, 2009.

[HCN09b] Miia Hermelin, Joo Yeon Cho, and Kaisa Nyberg. A new technique for multidimensional linear cryptanalysis with applications on reduced round Serpent. In Pil Joong Lee and Jung Hee Cheon, editors, *Information Security and Cryptology - ICISC 2008*, volume 5461 of *LNCS*, pages 383–398. Springer, 2009.

[HCN08] Miia Hermelin, Joo Yeon Cho, and Kaisa Nyberg. Multidimensional linear cryptanalysis of reduced round Serpent. In Yi Mu, Willy Susilo, and Jennifer Seberry, editors, *Information Security and Privacy - ACISP 2008*, volume 5107 of *LNCS*, pages 203–215. Springer, 2008.

[HCN09] Miia Hermelin, Joo Yeon Cho, and Kaisa Nyberg. Multidimensional extension of Matsui's algorithm 2. In Orr Dunkelman, editor, *Fast Software Encryption - FSE 2009*, volume 5665 of *LNCS*, pages 209–227. Springer, 2009.

[HCN09c] Miia Hermelin, Joo Yeon Cho, and Kaisa Nyberg. Statistical tests for key recovery using multidimensional extension of Matsui's algorithm 1. EUROCRYPT 2009 POSTERSESSION, 2009.

[Gér10] Benoît Gérard. *Cryptanalyses statistiques des algorithmes de chiffrement à clef secrète.* PhD thesis, Université Pierre et Marie Curie, Paris, 2010.

[BCQ04] Alex Biryukov, Christophe De Cannière, and Michaël Quisquater. On Multiple Linear Approximations. In *CRYPTO 2004*, LNCS, pages 1–22. Springer–Verlag, 2004.

[BSKR94] Jr. Burton S. Kaliski and M. J. B. Robshaw. Linear Cryptanalysis Using Multiple Approximations. In *CRYPTO '94*, LNCS, pages 26–39. Springer–Verlag, 1994.

[CT91] T.M. Cover and J.A. Thomas. *Information theory.* Wiley series in communications. Wiley, 1991.

[Dum00] I. Dumer. Ellipsoid lists and maximum-likelihood decoding. *IEEE Trans. Info. Theory*, 46(2):649–656, March 2000.

[JV03] P. Junod and S. Vaudenay. Optimal key ranking procedures in a statistical cryptanalysis. In *FSE 2003*, LNCS, pages 235–246. Springer–Verlag, 2003.

[LT07] Pierre Loidreau and Cédric Tavernier. An algorithm for finding linear approximations and its application to 8-round of DES, 2007.

[Mat94] Mitsuru Matsui. The First Experimental Cryptanalysis of the Data Encryption Standard. In *CRYPTO '94*, LNCS, pages 1–11. Springer–Verlag, 1994.

[MM92] Atsuhiro Yamagishi Mitsuru Matsui. A New Method for Known Plaintext Attack of FEAL Cipher. In *EUROCRYPT '92*, LNCS, page 81. Springer–Verlag, 1992.

[MPWW95] S. Murphy, F. Piper, M. Walker, and P. Wild. Likelihood estimation for block cipher keys, 1995.

[OA94] Kazuo Ohta and Kazumaro Aoki. Linear Cryptanalysis of the Fast Data Encipherment Algorithm. *LNCS*, 839:12–16, 1994.

[TCG91] Anne Tardy-Corfdir and Henri Gilbert. A Known Plaintext Attack of FEAL-4 and FEAL-6. In *CRYPTO '91*, LNCS, pages 172–181. Springer–Verlag, 1991.

[TSM94] Toshio Tokita, Tohru Sorimachi, and Mitsuru Matsui. Linear Cryptanalysis of LOKI and s2DES. In *ASIACRYPT '94*, LNCS, pages 293–303. Springer–Verlag, 1994.