An Analysis of the XSL Algorithm

Carlos Cid[1,⋆] and Gaëtan Leurent[2]

[1] Information Security Group,
Royal Holloway, University of London,
Egham, Surrey TW20 0EX, United Kingdom
carlos.cid@rhul.ac.uk
[2] École Normale Supérieure,
Département d'Informatique, 45 rue d'Ulm,
Paris 75230 Cedex 05, France
gaetan.leurent@ens.fr

Abstract. The XSL "algorithm" is a method for solving systems of multivariate polynomial equations based on the linearization method. It was proposed in 2002 as a dedicated method for exploiting the structure of some types of block ciphers, for example the AES and Serpent. Since its proposal, the potential for algebraic attacks against the AES has been the source of much speculation. Although it has attracted a lot of attention from the cryptographic community, currently very little is known about the effectiveness of the XSL algorithm. In this paper we present an analysis of the XSL algorithm, by giving a more concise description of the method and studying it from a more systematic point of view. We present strong evidence that, in its current form, the XSL algorithm does not provide an efficient method for solving the AES system of equations.

Keywords: XSL algorithm, T′ method, Linearization, AES.

1 Introduction

In 2002 Courtois and Pieprzyk showed that recovering an AES encryption key was equivalent to solving a large system of multivariate quadratic equations over a small finite field [10,11]. They exploited the fact that the only non-linear component of the cipher (the S-Box) is based on the inverse map over the finite field $\mathbb{F}_{2^8}$, and were able to obtain a set of multivariate quadratic equations that completely described the S-Box transformation. By combining all equations throughout the cipher, they were able to express the full encryption transformation as a large, sparse and overdefined system of multivariate quadratic equations over $\mathbb{F}_2$ (in total 8000 equations with 1600 variables for the AES with 128-bit keys).

The problem of solving systems of multivariate quadratic equations over a finite field is known to be NP-complete, and it is widely believed that the commonly applied techniques (such as Gröbner Basis algorithms) cannot generally be used for efficiently solving systems with more than a handful of variables. However the system derived from the AES is very structured, and the hope is that a

⋆ This author was supported by EPSRC Grant GR/S42637.

B. Roy (Ed.): ASIACRYPT 2005, LNCS 3788, pp. 333–352, 2005.

dedicated method can exploit this rich structure. With that in mind, a method called XSL was proposed in [10,11], which it was claimed could provide an efficient way to recover the encryption key for certain types of block ciphers. According to the estimates presented in [10], with the XSL algorithm one could mount a (at least theoretical) successful attack against the AES with 256-bit keys.

Around the same time, Murphy and Robshaw [13] showed how to express the AES encryption as a far simpler system of equations over $\mathbb{F}_{2^8}$. It was noticed then that, if XSL worked as predicted, this system should be easier to solve than the original one over $\mathbb{F}_2$, and in theory could provide an efficient attack against the AES with 128-bit keys [13,14].

Since the introduction of the XSL algorithm, the potential for algebraic attacks against block ciphers (and in particular the AES) has been the source of much speculation. Although it has attracted a lot of attention from the cryptographic community, currently very little is known about the effectiveness of the XSL algorithm, and of algebraic attacks in general, against block ciphers.

In this paper we present an analysis of the XSL algorithm. Based on our results we conclude that, as presented in [11], the XSL algorithm should not provide an efficient method for solving the AES system of equations.

2 Linearization Methods

The XSL algorithm was introduced in [10,11], and it is derived from an earlier algorithm called XL [8]. The XL algorithm and its many variants [7,9,11] are all based on the method of *linearization*, a well-known technique for solving large systems of multivariate polynomial equations. In this method we consider all monomials in the system as independent variables and try to solve it using linear algebra techniques. Note that the linearization method can only be successful if the number of *linearly independent* equations is approximately the same as the number of monomials in the system. The XL algorithm and its variants attempt to generate enough equations when this is not the case.

The XL is a simple algorithm: if we consider a system of m quadratic equations and n variables over a finite field $\mathbb{K}$,

$$f_1(x_1, \ldots, x_n) = 0 \ , \ \ldots \ , \ f_m(x_1, \ldots, x_n) = 0, \tag{1}$$

the algorithm simply multiplies the original equations by all monomials M_i up to a prescribed degree $D - 2$, and attempts to solve the system of all resulting equations

$$M_i \cdot f_j(x_1, \ldots, x_n) = 0 \tag{2}$$

of degree at most D by linearization.

Although not fully understood when first introduced, currently there seems to be a much better understanding of the behaviour of the XL algorithm, including its merits and limitations [1,2,3,4,12]. In particular it has been shown that some of the heuristics used in deriving the complexity of the XL algorithm [8] were too optimistic [12].

The XSL algorithm works slightly different. Whereas in the XL algorithm the equations are multiplied by all monomials up to a certain degree, in the XSL algorithm the equations are multiplied only by "carefully selected monomials". The goal here is to create fewer new monomials when generating the new equations. Additionally, there is a last step (called T′ method), in which we try to obtain new linearly independent equations without creating any new monomials.

Analysis of the XSL algorithm does not seem to be an easy task, and currently very little is known about its behaviour. There are a number of reasons for this. Firstly, XSL can be considered an *ad-hoc* method, and the algorithm relies on the system presenting a somewhat special form, such as having "S-Boxes" with overdefined system of equations, repeated layers of linear equations, and so on. Secondly there are different versions of the algorithm (two attacks are given in [10], which are substantially different from the attack proposed in [11]), and in all cases, the description given leaves some room for interpretation. Furthermore, given the size of the systems involved, it is very difficult to implement and run experiments even on small examples to verify the heuristics in [10,11].

In the following sections, we give a more concise description of the XSL algorithm and study it from a more systematic point of view in an attempt to get an insight into the algorithm and better understand its behaviour.

3 The XSL Algorithm

There are different versions of the XSL algorithm. The first version was proposed in [10], where two different attacks were described: the first one eliminating the key schedule equations (but requiring a number of plaintext-ciphertext pairs), and a second, more specific attack, that used the key schedule equations (and should work with a single plaintext-ciphertext pair). Later a different version of the algorithm was introduced in [11] (called "compact XSL"). Only the first attack was described in [11], although it is straightforward to extend the method to the second attack.

In this paper we concentrate on the "compact XSL" algorithm. Although the algorithm can in theory be applied to a number of block ciphers, our analysis is focused on the AES, and we take into account the special structure of the systems derived from this cipher. The systems used are over $\mathbb{F}_2$ and always include the key schedule equations (i.e. we perform the second XSL attack).

The XSL algorithm, as described in [11], is supposed to work only on special types of ciphers; it assumes that the cipher is built with layers of small S-Boxes interconnected by linear key-dependent layers. The S-Box is such that it can be described by an overdefined set of quadratic equations. To apply the second attack (i.e. including the key schedule), the key schedule needs to have a similar structure to the encryption (which is the case for the AES).

The XSL algorithm consists of four main steps:

1. Process the existing set of equations, by choosing certain sets of monomials and equations that will be used during the later steps of the algorithm.

2. Select the value of the parameter P, and multiply the chosen equations by the product of $P-1$ selected monomials. This is the "core" of the XSL attacks and should generate a large number of equations whose terms are the product of the monomials chosen earlier.
3. Perform the T$'$ method, in which some selected equations are multiplied by single variables. The goal is to generate new equations without creating any new monomials. Iterate with as many variables as necessary until the system has enough linearly independent equations to apply linearization[1].
4. Apply linearization, by considering each monomial as a new variable and performing Gaussian elimination. This should yield a solution for the system.

In the following sections we describe the first three steps, in an attempt to better understand the behaviour of the XSL algorithm. During our analysis, we illustrate the working of the algorithm on a small variant of the AES defined in [5]. The cipher used (denoted by SR(3,1,1,4)) has a 4-bit block and 3 rounds, and its operations are over the field $\mathbb{F}_{2^4}$. We note however that this small cipher is used only to assist the understanding of the algorithm's various steps; all results obtained are valid for the full AES, and we always present figures for this cipher. We use the following notation throughout this paper (similarly to [11]):

B: number of S-Boxes in each encryption round;	N_r: number of encryption rounds;
$\mathcal{R}$: set of all equations;	R: cardinality of $\mathcal{R}$;
$\mathcal{E}$: subset of $\mathcal{R}$ consisting of all L.I. equations;	E: cardinality of $\mathcal{E}$;
$\mathcal{T}$: set of all monomials in the system;	T: cardinality of $\mathcal{T}$;
$\mathcal{T}_i'$: set of monomials in the system such that $x_i \cdot \mathcal{T}_i' \subseteq \mathcal{T}$;	T': cardinality of $\mathcal{T}_i'$;
	s: number of bits on the S-Box;
t: number of monomials in the S-Box equations;	r: number of equations in an S-Box;
t_i': number of monomials in the S-Box equations to be used in the T$'$ method;	
L: number of subsets of linear layer equations;	S: total number of S-Boxes;
S_m: number of encryption S-Boxes;	S_k: number of key schedule S-Boxes;
b_i: number of neighbouring S-Boxes for equations in the subset i;	
N_b: number of columns in the data array;	N_a: number of rows in the data array.

4 Step 1 - Processing of the Original Set of Equations

The processing method suggested in [11] is that for every S-Box, a basis of $t-r$ monomials is chosen and the remaining r monomials are written as linear combinations of the elements of the basis. Furthermore, the basis should be chosen such that the variables (i.e. monomials of degree 1) are not in the basis, and the constant monomial 1 is in the basis.

For the AES, we have $r = 24$ and $t = 81$, so each S-Box has a basis consisting of 57 monomials. If we denote by w_{ij} and x_{ij} the j^{th} bit of the input and output of the i^{th} S-Box respectively, we can choose our basis such that it consists of the

[1] The T$'$ method has also been proposed as the final step of the XL algorithm, in the so-called XL2 method [9].

monomials $x_{ij}w_{ik}$, with $j \neq k$, and 1. In our small example, we have $r = 12$ and $t = 25$, so after this processing the S-Box equations would be given by

$$\begin{cases}
w_{10} & +w_{10}x_{11} + w_{11}x_{10} + w_{11}x_{12} + w_{12}x_{10} + w_{13}x_{11} + 1 \\
w_{11} & +w_{10}x_{11} + w_{10}x_{13} + w_{11}x_{13} + w_{12}x_{10} + w_{12}x_{13} + w_{13}x_{10} + w_{13}x_{11} \\
w_{12} & +w_{10}x_{11} + w_{10}x_{12} + w_{12}x_{11} + w_{12}x_{13} + w_{13}x_{10} + w_{13}x_{11} \\
w_{13} & +w_{10}x_{11} + w_{10}x_{12} + w_{10}x_{13} + w_{11}x_{10} + w_{11}x_{13} + w_{12}x_{10} + w_{12}x_{13} + w_{13}x_{10} \\
x_{10} & +w_{10}x_{11} + w_{10}x_{12} + w_{11}x_{10} + w_{11}x_{13} + w_{12}x_{11} + 1 \\
x_{11} & +w_{10}x_{12} + w_{10}x_{13} + w_{11}x_{10} + w_{11}x_{13} + w_{13}x_{10} + w_{13}x_{11} + w_{13}x_{12} \\
x_{12} & +w_{10}x_{13} + w_{11}x_{10} + w_{11}x_{12} + w_{11}x_{13} + w_{12}x_{10} + w_{13}x_{12} \\
x_{13} & +w_{10}x_{11} + w_{10}x_{12} + w_{10}x_{13} + w_{11}x_{10} + w_{12}x_{10} + w_{13}x_{10} + w_{13}x_{11} + w_{13}x_{12} \\
w_{10}x_{10} & +w_{10}x_{11} + w_{11}x_{10} + w_{12}x_{13} + w_{13}x_{12} + 1 \\
w_{11}x_{11} & +w_{10}x_{12} + w_{10}x_{13} + w_{11}x_{12} + w_{12}x_{10} + w_{12}x_{11} + w_{12}x_{13} + w_{13}x_{10} + w_{13}x_{12} \\
w_{12}x_{12} & +w_{10}x_{11} + w_{11}x_{10} + w_{11}x_{13} + w_{12}x_{13} + w_{13}x_{11} + w_{13}x_{12} \\
w_{13}x_{13} & +w_{10}x_{13} + w_{11}x_{12} + w_{12}x_{11} + w_{13}x_{10},
\end{cases}$$

and the basis would be given by

$$\{\, w_{10}x_{11}, w_{10}x_{12}, w_{10}x_{13}, w_{11}x_{10}, w_{11}x_{12}, w_{11}x_{13}, \\ w_{12}x_{10}, w_{12}x_{11}, w_{12}x_{13}, w_{13}x_{10}, w_{13}x_{11}, w_{13}x_{12}, 1 \,\}.$$

The set consisting of the monomials in the bases of all the S-Boxes is used to multiply the remaining equations in the system (the linear layer equations) in step 2 of the algorithm, while the S-Box relations are used to carry out substitutions in the linear layer equations (Section 5). One of the main ideas of the XSL algorithm is that during the attack the equations are always expressed as sum of terms that are the product of monomials in the bases of P different S-Boxes.

When performing the second XSL attack, we need to do the same processing with the key schedule S-Boxes. In this case we denote by k_{ij} and s_{ij} the j^{th} bit of the input and output of the i^{th} key schedule S-Box, respectively. Similarly to the encryption S-Boxes, we choose our basis such that it consists of the monomials $k_{ij}s_{ik}$, with $j \neq k$, and 1. We note however the key schedule has a slightly different structure from the encryption, such that not every key variable goes through an S-Box. The suggestion in [10] is that we should introduce the so-called "artificial S-Boxes", with the necessary variables and no equations. We find this a unnecessary and somewhat cumbersome step, which makes our analysis a bit more complex. In particular, it is harder to derive accurate figures for the number of monomials and equations in the resulting system. In our opinion it is better to rewrite the key schedule system such that these "artificial S-Boxes" are no longer required (see Appendix A). Either way, the chosen form for the key schedule equations should not be relevant in the analysis that follows and does not have any significant influence on the complexity of the attack described.

The linear layer equations (from the encryption and the key schedule) are the equations that will be used directly in step 2 of the algorithm. Each equation (called "active equation") will be multiplied by monomials of the basis from some $(P-1)$ different S-Boxes (called "passive S-Boxes"). The S-Box relations are not *explicitly* used in the algorithm, but rather in an indirect form. The linear layer equations are linear in the many variables of the system, and these

variables are not in the basis of any S-Box. Thus the XSL algorithm requires us to substitute the variables by their expressions as linear combination of the monomials from the corresponding S-Box basis prior to multiplication. Again, the idea of the XSL algorithm is that during the attack the equations are always expressed as sum of terms that are the product of monomials in the bases of the S-Boxes. For example, in our small cipher the initial key addition operation is expressed by the following subsystem:

$$\begin{cases} p_0 + w_{10} + k_{00} \\ p_1 + w_{11} + k_{01} \\ p_2 + w_{12} + k_{02} \\ p_3 + w_{13} + k_{03}, \end{cases} \tag{3}$$

where the p_i variables correspond to the plaintext values. After performing the substitution of the monomials w_{1j} and k_{0j} by their respective expressions from the corresponding S-Boxes bases, the subsystem (3) is written as:

$$\begin{cases} p_0 + w_{10}x_{11} + w_{11}x_{10} + w_{11}x_{12} + w_{12}x_{10} + w_{13}x_{11} + \\ \quad + k_{00}s_{01} + k_{01}s_{00} + k_{01}s_{02} + k_{02}s_{00} + k_{03}s_{01}, \\ p_1 + w_{10}x_{11} + w_{10}x_{13} + w_{11}x_{13} + w_{12}x_{10} + w_{12}x_{13} + w_{13}x_{10} + w_{13}x_{11} + \\ \quad + k_{00}s_{01} + k_{00}s_{03} + k_{01}s_{03} + k_{02}s_{00} + k_{02}s_{03} + k_{03}s_{00} + k_{03}s_{01}, \\ p_2 + w_{10}x_{11} + w_{10}x_{12} + w_{12}x_{11} + w_{12}x_{13} + w_{13}x_{10} + w_{13}x_{11} + \\ \quad + k_{00}s_{01} + k_{00}s_{02} + k_{02}s_{01} + k_{02}s_{03} + k_{03}s_{00} + k_{03}s_{01}, \\ p_3 + w_{10}x_{11} + w_{10}x_{12} + w_{10}x_{13} + w_{11}x_{10} + w_{11}x_{13} + w_{12}x_{10} + w_{12}x_{13} + w_{13}x_{10} + \\ \quad + k_{00}s_{01} + k_{00}s_{02} + k_{00}s_{03} + k_{01}s_{00} + k_{01}s_{03} + k_{02}s_{00} + k_{02}s_{03} + k_{03}s_{00}. \end{cases}$$

The processing above is performed on all equations arising from the linear layer system (including the key schedule). This results in $(N_r + 1) \cdot B \cdot s + K_e$ *quadratic* equations over $\mathbb{F}_2$, with $2s \cdot S$ variables and $S \cdot (t - r - 1)$ monomials (excluding the constant monomial), where K_e is the number of key schedule equations and S is the total number of S-Boxes in the cipher. In our small example $S = 6$ and $K_e = 8$, so we have $4 \cdot 1 \cdot 4 + 8 = 24$ equations on 48 variables and 72 monomials. For the AES-128, we have $S = 10 \cdot (16 + 4) = 200$ and $K_e = 192$. Thus there are 1600 equations, 3200 variables and 11200 monomials (Appendix A).

5 Step 2 - Multiplying the Equations

In this step, the attacker selects the value of the parameter P (refer to [11] on how to compute P), and then multiplies each of the equations derived from the cipher linear layer after the substitution described above by the product of $(P-1)$ monomials from different S-Boxes. Only the monomials in the bases are used. To ensure that the equations generated contain only terms that are the product of monomials from P *different* S-Boxes, a few neighbouring S-Boxes need to be excluded (i.e. S-Boxes that have monomials in common with the active equation). This can be visualised in the diagram illustrating the encryption operation in our small example (Figure 1). For example, when multiplying the equations in the subset Lin_2, we should not include the monomials in S-Boxes S_2, S_3 and K_2.

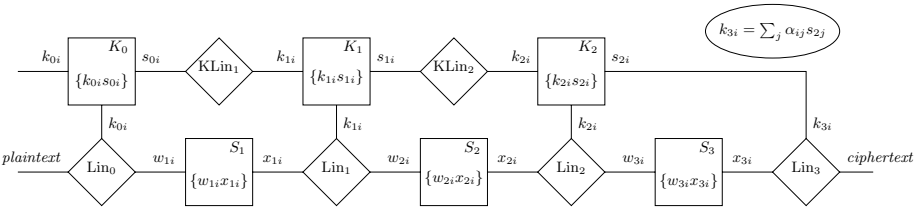

Fig. 1. S-Boxes and Linear Layers on the SR(3,1,1,4) encryption

After multiplication, we expect to have $R = \sum_{i=1}^{L} s \sum_{k=1}^{P} (t-r-1)^{k-1} \binom{S-b_i}{k-1}$ equations (though not all linearly independent), where L is the number of subsets of linear layer equations and b_i is the number of neighbouring S-Boxes for the subset i. In total, we expect to have $T = \sum_{k=0}^{P} (t-r-1)^k \binom{S}{k}$ monomials in the system (Appendix A).

As computed in the previous section, we have 1600 quadratic equations on 3200 variables and 11200 monomials for the AES-128 before multiplication[2]. So it appears that we start with an *underdefined* system, which in principle should not be solvable. Note however that, apart from the initial substitution, we have not used the S-Boxes relations yet.

It is not completely clear from the description in [11] how to include the S-Boxes equations. The authors say that "each time, in the attack we want to use one of the other r terms [not in the S-Box basis], we will write them as linear combination of the elements of the basis" [11]. Although this description leaves the method somewhat open for interpretation, we believe that the most likely way to proceed is to generate all equations via multiplication and then perform (as much as possible) substitutions of monomials not in the bases by their expressions with the corresponding linear combination of monomials in the basis. This should hopefully introduce many new equations. Note that because the initial system used by the XSL algorithm is *underdefined*, the system can only be solved if further substitutions are performed.

As before, let w_{ij} and x_{ij} be the j^{th} bit of the input and output of the i^{th} S-Box respectively, such that the basis consists of the monomials $x_{ij} w_{ik}$, with $j \neq k$, and 1 (note that on the key schedule S-Boxes, the variables should be k_{ij} and s_{ij}, but for simplicity we rename these variables). We denote by $[w_{ij}]$, $[x_{ij}]$ and $[x_{ij} w_{ij}]$ the expressions of these monomials as linear combination of the monomials in the S-Box basis. When performing substitutions, we need to make sure that variables are always substituted in pairs, from the same S-Box (w_{ij} and x_{ik}). This is required to ensure that the resulting new equations are still made up of terms that are the product of monomials from the bases of the S-Boxes. Furthermore, we should also make sure that the substitutions do not create monomials of degree higher than $2P$.

[2] Appendix A of [11] describes how to simplify the equations and reduce the number of variables. However this new format does not seem to be suitable for the XSL attack.

The relations used for substitution and generation of new equations are

$$\begin{aligned}
(x_{ij}w_{ik}) \cdot (x_{ij}w_{ik}) &= x_{ij}w_{ik} && \text{for any } i,j,k \\
(x_{ij}w_{ik}) \cdot (x_{ij}w_{il}) &= (x_{ij}w_{ik}) \cdot [w_{il}] = [w_{ik}] \cdot (x_{ij}w_{il}) && \text{for any } i,j,k \\
(x_{ij}w_{ik}) \cdot (x_{il}w_{ik}) &= (x_{ij}w_{ik}) \cdot [x_{il}] = [x_{ij}] \cdot (x_{il}w_{ik}) && \text{for any } i,j,k \\
x_{ij}w_{ik} &= [x_{ij}] \cdot [w_{ik}] && \text{for } j \neq k \\
x_{ij}w_{ik} &= [x_{ij}] \cdot [w_{ik}] = [x_{ij}w_{ik}] && \text{for } j = k.
\end{aligned} \tag{4}$$

For each S-Box, the number of relations is $s^2+s^3+s^3+s(s-1)+2s = 2s^3+2s^2+s$.

Note that substitutions using any of the relations in (4) will always result in (or only be possible by) monomials made up of the product of some monomials from the *same* S-Box. However, the XSL algorithm described in [11] excludes monomials from neighbouring S-Boxes when multiplying the original equations, and so the generated equations have only terms of the form

$$x_{i_1 j_1} w_{i_1 k_1} \cdot x_{i_2 j_2} w_{i_2 k_2} \cdot \ldots \cdot x_{i_l j_l} w_{i_l k_l}, \tag{5}$$

with $l \leq P$ and all i_r's pairwise distinct. This means that no substitutions can be made such that the resulting new equations contain only terms that are the product of up to P monomials from *different* S-Boxes. Substitutions always introduce new monomials, and this is not intended to happen with the XSL algorithm. Without any substitutions, we never get any new expressions, and the method essentially ignores the S-Box equations. Therefore, no matter how large the parameter P is, there is no hope that the XSL algorithm (as described in [11]) can solve the initial set of equations[3].

The problem with the XSL algorithm arises from the attempt to have only monomials made up of the product of P *different* S-Boxes, and as such some S-Boxes needed to be excluded when multiplying. The simplest way to get round this situation is to allow the product of any P monomials from the bases, not necessarily from different S-Boxes, and use all S-Boxes when multiplying, including the neighbouring ones. The effect is that we should expect a larger number of monomials in the end (as well as equations), but this will also allow the substitutions, and we will be able to include the S-Boxes relations in the computations.

A more systematic way to proceed is however to add the relations (that were to be used for substitution) to the initial set of equations, and perform the algorithm without any further substitutions. Care has to be taken though, as some of the new equations have degree 4 rather than 2 (e.g. $x_{ij}w_{ik} = [x_{ij}] \cdot [w_{ik}]$), and these should be multiplied by the product of up to $P-2$ monomials only. We note also that, as the monomial $x_{ij}w_{ij}$ does not belong to the S-Box basis, we should not include some of the relations involving this monomial (for example, $x_{ij}w_{ij} = [x_{ij}w_{ij}]$) in the initial set of equations.

It can be shown that this new procedure is essentially equivalent to the previous one, and all new equations created by substitution can also be generated by applying the method to this enlarged set of equations. We call this modified method *sXL* (standing for *substitute and XL*), and examine it in the following section.

[3] Substitutions could still be performed by modifying the last step of XSL (T′ method), but this is obviously not the way it was originally proposed.

5.1 The sXL Algorithm

The sXL algorithm seems to be the natural way to get round the flaw in the original XSL algorithm described in [11]. In the sXL algorithm, equations are first processed as described in Section 4. We then add the many new relations (4) resulting from the S-Boxes equations to the original linear layer equations, and multiply all equations in this set by the product of $(P-1)$ monomials from the bases of (not necessarily distinct) S-Boxes, for an appropriate value P.

In the initial set, there were $(N_r+1)\cdot B\cdot s+K_e$ quadratic equations on $2s\cdot S$ variables and $S\cdot(t-r-1)$ monomials. To this set we add

$$S\cdot(s(s-1)+s(s-1)^2+s(s-1)^2+s(s-1)+s)=S\cdot(2s^3-2s^2+s)$$

quartic equations derived from the relations in (4) (we are excluding some relations using the monomial $x_{ij}w_{ij}$). We call this new set $\overline{\mathcal{S}}$.

To analyse the running time of the sXL algorithm, we need to compute the minimal value P_m of the parameter P for which the method yields a solution of the system. We initially ignore the T′ method (Section 6).

In order to compute P_m, we introduce new variables Y_{ijk} and substitute the monomials $(x_{ij}\cdot w_{ik})$ in the equations in $\overline{\mathcal{S}}$ by Y_{ijk}. We denote the resulting new set of equations by $\mathcal{S}\subset\mathbb{K}[Y]$. The new variables Y_{ijk} are related by the various relations of type

$$Y_{ijk}\cdot Y_{ipq}=x_{ij}w_{ik}\cdot x_{ip}w_{iq}=x_{ij}w_{iq}\cdot x_{ip}w_{ik}=Y_{ijq}\cdot Y_{ipk}, \tag{6}$$

where we might have to use the S-Box relations if $j=q$ or $p=k$. We call this set $\mathcal{R}\subset\mathbb{K}[Y]$, and it contains $S\cdot\frac{s^2(s-1)^2}{4}$ equations.

We now consider the system of equations $\mathcal{S}\cup\mathcal{R}\subset\mathbb{K}[Y]$, and execute the XL algorithm on this system. The algorithm is required to run to a certain degree D_m to yield a solution.

We now have the following proposition (proof is given in Appendix B):

Proposition 1. *Let $\overline{\mathcal{S}}$ be the set consisting of the original linear layer equations together with the relations (4) resulting from the S-Boxes equations, all written as sum of terms made up of the product of monomials in the S-Boxes bases. Denote by P_m the minimal value of the parameter P for which the algorithm described above (sXL) yields a solution of the system. Similarly, let $\mathcal{S}\cup\mathcal{R}\subset\mathbb{K}[Y]$ denote the set of equations derived from $\overline{\mathcal{S}}$ and the relations (6) by substituting the monomials $(x_{ij}\cdot w_{ik})$ by Y_{ijk}. If D_m denotes the minimal degree for which the XL algorithm yields a solution of this system, then $P_m=D_m$.*

Proposition 1 states that the sXL algorithm is essentially equivalent to an initial substitution (substituting the monomials $(x_{ij}\cdot w_{ik})$ by Y_{ijk}), and then applying the XL algorithm to the resulting system in $\mathbb{K}[Y]$ (thus the name *sXL - substitute and XL*). For the AES-128, we start the XSL algorithm with 1600 equations, 3200 variables and 11200 monomials (i.e. an underdefined system). To run the sXL algorithm, we use the set $\overline{\mathcal{S}}$, which contains 182400 linearly independent equations. The set $\mathcal{R}$ has 156800 linearly independent equations, and after adding

all relations and substituting the monomials, the set $\mathcal{S} \cup \mathcal{R}$ has 276800 equations (each S-Box contains 1376 linearly independent equations) on 11200 variables. By Proposition 1 above and Theorem 1 from [12], we expect to run the algorithm up to degree at least $D = 51$ for the method to yield a solution. If we include the T′ method as last stage (essentially running the XL2 method [9]), we expect to run the algorithm to degree at least $D = 20$. Thus in the best case, the complexity of the attack is at least

$$\begin{aligned}(\dim(\phi(\overline{U}_D)))^\omega &= (\dim(U_D) - \dim(\ker\phi))^\omega \\ &\geq \left(\sum_{i=0}^{20}\binom{11200}{i} - 156800 \cdot \sum_{i=0}^{18}\binom{11200}{i}\right)^\omega \approx 2^{492},\end{aligned}$$

where ϕ, U_D, $\overline{U}_D$ are defined in the proof of Proposition 1 (Appendix B), and $\omega = 2.376$ is the highly optimistic Gaussian reduction exponent given in [11]. Furthermore it should be clear that there seems to be no benefit in running this method instead of simply applying XL or XL2 to the simplified AES system of 8000 equations over 1600 variables described in [10]. Using the same results from [12], we expect in this case to run the algorithm up to degree at least $D = 44$ for the XL algorithm and at least $D = 29$ for the XL2 method. Again, in the best case the complexity of the attack is at least

$$T^\omega = \left(\sum_{i=0}^{29}\binom{1600}{i}\right)^\omega \approx 2^{488}.$$

We recall that the inefficiency of the XL algorithm against the AES has already been shown in [11], and this was in fact the motivation for the proposal of the XSL algorithm. We have shown however that the XSL algorithm presented in [11] has a flaw in its description, and the natural modification (i.e. sXL) is essentially equivalent to the XL algorithm (or XL2) on a much larger system, resulting therefore in a less efficient method of attack against the AES.

6 Step 3 - The T′ method

The T′ method is the final stage of the XSL algorithm before linearization. We recall that to apply linearization, we require that the number of *linearly independent* equations in the system needs to be approximately the same as the number of monomials (in the notation introduced earlier, $E \approx T$). Starting with a system resulting from step 2 (which may still have T much larger than E), the T′ method works by multiplying some selected equations by single variables x_i (reducing modulo $x_i^2 + x_i$ when necessary) in an attempt to obtain new linearly independent equations without creating any new monomials. The hope is that after a few iterations we have $E = T - 1$. Although the method seems to have been designed to work on systems of equations over $\mathbb{F}_2$, it is possible to modify it to work on equations over other finite fields.

Let $\mathcal{R}$ be a system of multivariate polynomial equations of degree at most D with n variables $\{x_1, x_2, \ldots, x_n\}$ over the finite field $\mathbb{K} = \mathbb{F}_2$. We assume that

$\mathcal{R}$ contains E linearly independent equations. Let $\mathcal{T}$ be the set of all monomials in the system, and $\mathcal{T}'_i$ be the set of monomials that can be multiplied by the variable x_i and still belong to $\mathcal{T}$, i.e. $\mathcal{T}'_i = \{t \in \mathcal{T} | x_i \cdot t \in \mathcal{T}\}$.

Denote by T and T'_i the cardinality of the sets $\mathcal{T}$ and $\mathcal{T}'_i$, respectively. Assuming that $E \geq T - T'_i + C$ and $C \geq 1$, we can apply the following "algorithm" [11].

1. Perform a Gaussian elimination on the system $\mathcal{R}$ to bring it to a form in which each monomial is a known linear combination of monomials in $\mathcal{T}'_i$. Since we have $E \geq T - T'_i + C$, we should have around C equations of which all monomial are in $\mathcal{T}'_i$.
2. Multiply these equations by x_i, reducing modulo $x_i^2 + x_i$ when necessary. Add any new linearly independent equations to the system $\mathcal{R}$.
3. Repeat steps 1 and 2 on the resulting system with other variables x_j until $E = T - 1$.

It is expected in [11] that the number of new equations generated grows at exponential rate, and that if the initial system has a unique solution, then after a few iterations (perhaps using as little as three variables) the algorithm should generate enough equations to solve the system by linearization.

Consider the polynomials in $\mathcal{R}$ as vectors over $\mathbb{K}$ in the polynomial algebra $\mathbb{K}[x_1, \ldots, x_n]$ and $\mathcal{E}$ the vector space (of dimension E) generated by $\mathcal{R}$. With an abuse of notation, we denote the space generated by all monomials of degree at most D by $\mathcal{T}$. By using the field relation $x^2 + x = 0$ to reduce the degree of monomials when necessary, we have $T = \dim(\mathcal{T}) = \sum_{i=0}^{D} \binom{n}{i}$.

For any variable x_i, let $\mathcal{T}'_i \subseteq \mathcal{T}$ be the subspace of $\mathcal{T}$ defined earlier. We can write

$$\mathcal{T} = \mathcal{T}'_i \oplus \mathcal{U} \quad \text{and} \quad T'_i = \dim(\mathcal{T}'_i) = \sum_{i=0}^{D-1} \binom{n}{i} + \binom{n-1}{D-1}. \tag{7}$$

In order to apply the T$'$ method, we need $\mathcal{E} \cap \mathcal{T}'_i \neq \emptyset$. The vectors in $\mathcal{E} \cap \mathcal{T}'_i$ correspond to the equations that are multiplied by the variable x_i when running the algorithm. A sufficient condition is that

$$\dim(\mathcal{E}) > \dim(\mathcal{U}) = \dim(\mathcal{T}) - \dim(\mathcal{T}'_i), \tag{8}$$

or equivalently, that $E > T - T'_i$. We denote the subspace $\mathcal{E} \cap \mathcal{T}'_i$ by $\mathcal{C}_i$, and its dimension by $C_i = E - T + T'_i$.

We note that the multiplication of the equations in $\mathcal{E} \cap \mathcal{T}'_i$ by x_i induces a linear transformation $X_i : \mathcal{T}'_i \to \mathcal{T}'_i$. By appropriately choosing an ordered basis for $\mathcal{T}'_i$, X_i can be represented by the $T' \times T'$ matrix $\begin{pmatrix} 0 & 0 \\ 0 & \mathrm{Id} \end{pmatrix}$, where Id corresponds to the $\frac{T'_i}{2} \times \frac{T'_i}{2}$ identity matrix. The image of X_i is generated by $\{x_i, x_1x_i, x_2x_i, \ldots, x_1 \ldots x_i \ldots x_n\}$. The T$'$ method simply computes $X_i(\mathcal{C}_i)$ and adds the resulting vectors to the space $\mathcal{E}$. If we denote by η_k the number of new equations generated by the k^{th} iteration of the algorithm using the variable x_{i_k}, then

$$\eta_k \leq \min(\gamma + \eta_{k-1}, \dim(\mathrm{Im}(X_{i_k}))), \tag{9}$$

where $\gamma = E - T + T'$ for the initial system if x_{i_k} is a new variable, otherwise $\gamma = 0$. This shows that the number of new equations generated by the method does not grow at exponential rate as suggested in [11].

It should be clear that if $X_i(\mathcal{C}_i) \subseteq \mathcal{C}_i$, then the T$'$ method applied to the variable x_i in a particular iteration of the algorithm does not generate any new linearly independent equations. We should then try other variables, as suggested in [11], in the hope that new equations are generated. These could be then added to the system, and the process could be repeated with further variables (including x_i). However, once the condition above is met by all variables, no new equations can be generated. Thus we have the following lemma.

Lemma 1. *Let $\mathcal{R}$ be a system of m multivariate equations of degree $D \geq 2$ with n variables $\{x_1, x_2, \ldots, x_n\}$ over the finite field $\mathbb{K} = \mathbb{F}_2$, and let $\mathcal{C}_i$ and X_i be the $\mathbb{K}$-subspace of $\mathbb{K}[x_1, \ldots, x_n]$ and the linear transformation with respect to the variable x_i, as defined above. If $X_i(\mathcal{C}_i) \subseteq \mathcal{C}_i$ for every $1 \leq i \leq n$, then the* T$'$ *method does not generate any new linearly independent equation.*

Therefore if a system satisfies the conditions of Lemma 1 before we have enough linearly independent equations to apply linearization, the T$'$ method surely fails. Although it is not clear how likely a system is to satisfy these conditions, in Appendix C we present an example of a small system for which the T$'$ method does not work.

We can make some further remarks about the T$'$ method when it is applied as the final step for XL-type algorithms. Suppose that S is the initial system of m quadratic equations with n variables over the finite field $\mathbb{F}_2$. The XL algorithm multiplies these equations by all monomials up to a prescribed degree $d = D-2$, obtaining a much larger system $\mathcal{R}$ with $R = \sum_{i=0}^{D-2} \binom{n}{i} \cdot m$ equations. We expect to have

$$T = \sum_{i=0}^{D} \binom{n}{i} \quad \text{and} \quad T'_i = \sum_{i=0}^{D-1} \binom{n}{i} + \binom{n-1}{D-1}, \tag{10}$$

and therefore $T - T'_i = \binom{n-1}{D}$. The T$'$ method is supposed to work as soon as the number of linearly independent equations (E) is larger than $T - T'_i$. By the results of [12], we see that this condition can only be satisfied if T' is greater-or-equal to the coefficient of the D^{th} term of the expected Hilbert Series of a generic algebra of type $(n+1; m; d_1, \ldots, d_m)$.

Furthermore, given a variable x_i, the set $\mathcal{R}$ of equations can be divided into three subsets: (a) all equations obtained by multiplying monomials of degree up to $d - 1 = D - 3$, (b) all equations obtained by multiplying monomials of degree $d = D-2$ with the variable x_i, and (c) equations obtained by multiplying monomials of degree $d = D - 2$ without x_i. Thus we can write

$$R = \sum_{i=0}^{D-2} \binom{n}{i} \cdot m = \left(\sum_{i=0}^{D-3} \binom{n}{i} + \binom{n-1}{D-3} + \binom{n-1}{D-2} \right) \cdot m \tag{11}$$

To apply the T′ method, we should first perform a Gaussian reduction on the set $\mathcal{R}$, and then multiply the equations in $\mathcal{T}_i'$ by the variable x_i in an attempt to obtain new linearly independent equations.

It is clear that all equations in (a) and (b) are in $\mathcal{T}_i'$. However, the equations in (b) are fixed by x_i and no new equations will be generated by multiplication. For equations in (a), any new equations would have been already included when running the XL algorithm, so no new linearly independent equations can be generated by multiplication either.

The only useful equations of $\mathcal{R}$ for the T′ method are therefore the ones in (c), and the method can work if applied to (at most) $\binom{n-1}{D-2} \cdot m$ equations. This fact had already been remarked in [6].

In [15] it is shown how the T′ method can be interpreted in terms of Buchberger's Gröbner Basis algorithm. The method is further discussed (in the context of the XL2 [9] algorithm) in [2,4], where some doubts are cast on the general applicability of the method. It is remarked that the T′ method may not be able to run because some of the monomials in $\mathcal{T} \setminus \mathcal{T}'$ cannot be expressed as linear combination of monomials in $\mathcal{T}'$ (and therefore cannot be reduced). In particular, this will happen if $C = E - T + T'$ is small, because as we saw above, after the XL algorithm many equations are already in $\mathcal{T}'$.

It is also noted in [2] that the method should operate with all variables instead of just two or three. In this case the XL2 method is equivalent to running the XL algorithm one degree higher and eliminating all the highest degree monomials. However it is not hard to construct examples where two variables prove to be enough.

The T′ method is perhaps the least understood part of XL-type algorithms. Experiments have proved to be inconclusive, and more study may be needed to verify whether it can be used in general as a final step of algorithms for efficiently solving systems of multivariate equations.

7 Conclusion

Since the proposal of the XSL algorithm, the potential for algebraic attacks against block ciphers, and in particular the AES, has been the source of much speculation and has attracted a lot of attention from the cryptographic community. Although not much is known about the effectiveness of algebraic attacks as a cryptanalytic technique, it is widely believed that the most promising approach is the development of *dedicated* methods for specific block ciphers. The XSL algorithm is perhaps the first attempt to exploit the particular structure of the AES system of equations. We have shown however that, as presented in [11], the XSL algorithm cannot solve the system arising from the AES. By discussing some alternatives for the algorithm, we come to the conclusion that, in its current form, it is unlikely that the algorithm can provide an efficient method for solving the AES system of equations.

References

1. Gwénolé Ars, Jean-Charles Faugère, Hideki Imai, Mitsuru Kawazoe, and Makoto Sugita. Comparison Between XL and Gröbner Basis Algorithms. In Pil Joong Lee, editor, *Advances in Cryptology - ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 338–353. Springer, 2004.
2. Jiun-Ming Chen, Nicolas Courtois, and Bo-Yin Yang. On Asymptotic Security Estimates in XL and Gröbner Bases-Related Algebraic Cryptanalysis. In *ICICS*, volume 3269 of *Lecture Notes in Computer Science*, pages 401–413. Springer, 2004.
3. Jiun-Ming Chen and Bo-Yin Yang. All in the XL Family: Theory and Practice. In *Proceedings of the 7th International Conference on Information Security and Cryptology*, volume 3506 of *Lecture Notes in Computer Science*, pages 67–86. Springer, 2004.
4. Jiun-Ming Chen and Bo-Yin Yang. Theoretical Analysis of XL over Small Fields. In *Proceedings of the 9th Australasian Conference on Information Security and Privacy*, volume 3108 of *Lecture Notes in Computer Science*, pages 277–288. Springer, 2004.
5. Carlos Cid, Sean Murphy, and Matthew Robshaw. Small Scale Variants of the AES. In H. Gilbert and H. Handschuh, editors, *Fast Software Encryption - FSE 2005*, volume 3557 of *Lecture Notes in Computer Science*, pages 145–162. Springer-Verlag, 2005.
6. Don Coppersmith. Comments on Crypto-Gram Newsletter. `http://www.schneier.com/crypto-gram-0210.html`, October 2002.
7. Nicolas Courtois. Algebraic Attacks over $GF(2^k)$, Applications to HFE Challenge 2 and Sflash-v2. In F. Bao et al., editor, *PKC 2004*, volume 2947 of *LNCS*, pages 201–217. Springer-Verlag, 2004.
8. Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In Bart Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 392–407. Springer, 2000.
9. Nicolas Courtois and Jacques Patarin. About the XL algorithm ober GF(2). In M. Joye, editor, *Progress in Cryptology - CT-RSA 2003*, pages 140–156. Springer-Verlag, 2003.
10. Nicolas Courtois and Josef Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. Cryptology ePrint Archive, Report 2002/044, 2002.
11. Nicolas Courtois and Josef Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In Yuliang Zheng, editor, *Advances in Cryptology - ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 267–287. Springer, 2002.
12. Claus Diem. The XL-Algorithm and a Conjecture from Commutative Algebra. In Pil Joong Lee, editor, *Advances in Cryptology - ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 323–337. Springer, 2004.
13. Sean Murphy and Matthew Robshaw. Essential Algebraic Structure within the AES. In M. Yung, editor, *Advances in Cryptology - CRYPTO 2002*, volume 2442 of *LNCS*, pages 1–16. Springer-Verlag, 2002.
14. Sean Murphy and Matthew Robshaw. Comments on the Security of the AES and the XSL Technique. *Electronic Letters*, 39:26–38, 2003.
15. Makoto Sugita, Mitsuru Kawazoe, and Hideki Imai. Relation between XL algorithm and Gröbner Bases Algorithms. Cryptology ePrint Archive, Report 2004/112, 2004.

A The XSL Attack on the AES-128

In this Appendix we make some computations concerning the XSL attack against the AES with 128-bit keys.

A.1 Key Schedule

The AES key schedule presents a different structure from the encryption, in that not all key variables go through an S-Box. The suggestion in [10] is that, when performing the second XSL attack, one should introduce the so-called "artificial S-Boxes", with some key variables and no equations. Instead of that, in our analysis we rewrite the key schedule system such that these "artificial S-Boxes" are no longer required.

There are $S_k = N_a N_r$ S-Boxes in the AES key schedule, and a total of $sN_aN_b(N_r+1)$ subkeys variables, of which sN_aN_r go through an S-Box during the key schedule. So we choose to introduce sN_aN_r new variables, to represent the bits of the S-Box output $s_{j,3,i}$. For the AES-128, we have $N_a = N_b = 4$, $N_r = 10$, and so $S_k = 40$. A diagram for the key schedule of the AES-128 is shown in Figure 2.

The key schedule set of equations used in the XSL attack consists initially of $sN_aN_bN_r$ linear equations. We can however express all subkeys variables as linear expression of the $2sN_aN_r$ S-Boxes variables (representing the bits of $k_{j,3,i}$ and $s_{j,3,i}$), as shown in the equations below:

$$
\begin{aligned}
k_{0,0,i} &= k_{0,3,i} + k_{1,3,i} + k_{2,3,i} + k_{3,3,i} + s_{2,3,i} + s_{1,3,i} + s_{0,3,i} \\
k_{1,0,i} &= k_{0,3,i} + k_{1,3,i} + k_{2,3,i} + k_{3,3,i} + s_{2,3,i} + s_{1,3,i} \\
k_{2,0,i} &= k_{0,3,i} + k_{1,3,i} + k_{2,3,i} + k_{3,3,i} + s_{2,3,i} \\
k_{j,0,i} &= k_{j,3,i} + k_{j-1,3,i} + k_{j-2,3,i} + k_{j-3,3,i} && \text{for } j = 3 \ldots (N_r - 1) \\
k_{0,1,i} &= k_{0,3,i} + k_{2,3,i} + s_{1,3,i} \\
k_{1,1,i} &= k_{1,3,i} + k_{3,3,i} + s_{2,3,i} \\
k_{j,1,i} &= k_{j,3,i} + k_{j-2,3,i} && \text{for } j = 2 \ldots (N_r - 1) \\
k_{0,2,i} &= k_{0,3,i} + k_{3,3,i} + s_{2,3,i} \\
k_{j,2,i} &= k_{j,3,i} + k_{j-1,3,i} && \text{for } j = 1 \ldots (N_r - 1) \\
k_{N_r,0,i} &= k_{N_r-4,3,i} + k_{N_r-3,3,i} + k_{N_r-2,3,i} + k_{N_r-1,3,i} + s_{N_r-1,3,i} \\
k_{N_r,1,i} &= k_{N_r-4,3,i} + k_{N_r-2,3,i} + s_{N_r-1,3,i} \\
k_{N_r,2,i} &= k_{N_r-4,3,i} + k_{N_r-1,3,i} + s_{N_r-1,3,i} \\
k_{N_r,3,i} &= k_{N_r-4,3,i} + s_{N_r-1,3,i}
\end{aligned}
$$

The equations above can also be used to simplify the key schedule linear layer equations relating variables from S-Boxes. These equations can be written as

$$k_{j,3,i} = k_{j+4,3,i} + s_{j+3,3,i} \quad \text{for } j = 0 \ldots (N_r - 5). \tag{12}$$

We therefore have $N_a(N_r-4)$ sets of s linear equations, and so $K_e = N_a(N_r-4)s$. For the AES-128, we have $K_e = 192$. The number of key schedule S-Boxes needed to express the different subkeys is given in Table 1.

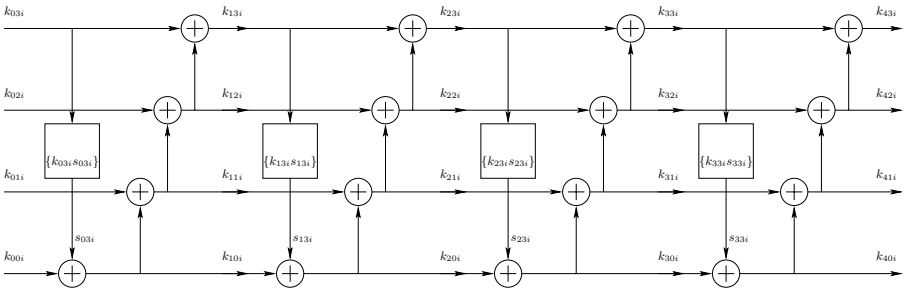

Fig. 2. Diagram for the AES-128 key schedule

Table 1. Number of S-Boxes used in equations involving $k_{j,r,i}$

j	0	1	2	3	4	5	6	7	8	9	10
$k_{j,0,i}$	4	4	4	4	4	4	4	4	4	4	4
$k_{j,1,i}$	3	3	2	2	2	2	2	2	2	2	3
$k_{j,2,i}$	3	2	2	2	2	2	2	2	2	2	2
$k_{j,3,i}$	1	1	1	1	1	1	1	1	1	1	2

A.2 Complexity of the XSL Attack on the AES-128

In this section we show that, in addition to the issues raised in Section 5, the XSL heuristics presented in [11] overestimate the number of equations generated by the algorithm[4]. Firstly, when deriving the complexity of the attacks, the XSL heuristics assume that all equations generated by the method are linearly independent. It should be clear that they are not. Even for $P = 2$, there are many relations of the type $f_i \cdot [f_j] = f_j \cdot [f_i]$. Secondly, the XSL algorithm states that neighbouring S-Boxes need to be excluded when multiplying the linear layer equations. This also needs to be taken into account when estimating the total number of equations.

The subsets of linear layer equations from the encryption have common variables with four S-Boxes from the current round, one S-Box from the next round (except in the first and last rounds, where some monomials are replaced by the plaintext or the ciphertext), and a number of key schedule S-Boxes. The number of neighbouring S-Boxes for the key schedule equations can be derived from Table 1, while the number of neighbouring S-Boxes for the encryption equations is given in Table 2.
Therefore the number of equations obtained by multiplication should be

$$R = \sum_{i=1}^{L} s \sum_{k=1}^{P} (t - r - 1)^{k-1} \binom{S - b_i}{k - 1} \tag{13}$$

[4] Note that although the key schedule equations were not used in [11], the way the heuristics were used to obtain the number of equations can be easily applied to the system including the key schedule.

Table 2. Number of neighbouring S-Boxes for the encryption equations (defining $w_{j,k,i}$)

j	0	1	2	3	4	5	6	7	8	9	10
$w_{j,0,i}$	5	9	9	9	9	9	9	9	9	9	8
$w_{j,1,i}$	4	8	7	7	7	7	7	7	7	7	7
$w_{j,2,i}$	4	7	7	7	7	7	7	7	7	7	6
$w_{j,3,i}$	2	6	6	6	6	6	6	6	6	6	6

instead of $Ss(t-r)^{(P-1)}\binom{S}{P-1}$ given in [11]. Likewise, the number of monomials is

$$T = \sum_{k=0}^{P} (t-r-1)^k \binom{S}{k} \tag{14}$$

instead of $(t-r)^P\binom{S}{P}$ given in [11]. For the AES-128, we have

$$S = S_m + S_k = N_a N_b N_r + N_a N_r = 200,$$
$$L = N_a N_b (N_r + 1) + N_a (N_r - N_b) = 200,$$

while b_i can be obtained from Tables 1 and 2.

Using these figures and the formulas given in [11], we obtain $P = 9$, giving $T \approx 2^{100}$ and $T^\omega \approx 2^{238}$ for the second XSL attack against the AES-128. We note however that we are not taking into account the linear dependencies between these equations, and so the complexity is likely to be much higher.

We also note that, with these new figures and assuming that *almost* all R equations are linearly independent [11], the T$'$ method seems to be irrelevant for the attack. In fact, since $T \approx 100T'$, when $P = 9$ we already have $R > T - 2$ (so there is no need for the T$'$ method), while for $P = 8$ we are still in the situation that $R < T - T'$ (and are therefore unlikely to be able to use the T$'$ method).

B Relation Between sXL and XL

We present here the proof of Proposition 1 from Section 5.1.

Let $\overline{\mathcal{S}}$ be the set of equations consisting of the original linear layer equations (after the processing described in Section 4), and the relations (4) resulting from the S-Boxes equations. All these equations are written as sum of terms made up of the product of monomials in the bases of the S-Boxes.

Let $D \in \mathbb{N}$ and $\overline{U}_D$ be the set of equations generated by running the sXL algorithm with the parameter $P = D$ on the set $\overline{\mathcal{S}}$. Denote by $\mathbb{K}[\{x_{ij} \cdot w_{ik}\}]$ the subring of $\mathbb{K}[x, w]$ generated by the various monomials of type $(x_{ij} \cdot w_{ik})$ contained in the bases of the S-Boxes. Furthermore, let $\mathbb{K}[\{x_{ij} \cdot w_{ik}\}]_{\leq 2D}$ and $\mathbb{K}[x, w]_{\leq 2D}$ be the $\mathbb{K}$-vector spaces generated by the respective polynomials of total degree at most $2D$. It is clear that we have $\overline{U}_D \subset \mathbb{K}[\{x_{ij} \cdot w_{ik}\}]_{\leq 2D}$.

Similarly to [12], we define

$$\overline{\chi}(D) = \dim_{\mathbb{K}}(\mathbb{K}[\{x_{ij} \cdot w_{ik}\}]_{\leq 2D}) - \dim_{\mathbb{K}}(\overline{U}_D).$$

The sXL algorithm will yield a solution for the system if $\overline{\chi}(D) = 1$ (we are ignoring by now the T′ method)[5]. We denote by P_m the minimal value of D for which this relation is satisfied.

We now introduce new variables Y_{ijk} and substitute the monomials $(x_{ij} \cdot w_{ik})$ in the equations in $\overline{\mathcal{S}}$ by Y_{ijk}. As the equations in $\overline{\mathcal{S}}$ are either quadratic or quartic, this can be done in a straightforward way. We denote this new set of equations by $\mathcal{S} \subset \mathbb{K}[Y]$. To this set we add the equations (6)

$$Y_{ijk} \cdot Y_{ipq} = Y_{ijq} \cdot Y_{ipk}, \tag{15}$$

contained in the set $\mathcal{R} \subset \mathbb{K}[Y]$. Let U_D be the set of equations generated by running the XL algorithm up to degree D on the set $\mathcal{S} \cup \mathcal{R} \subset \mathbb{K}[Y]$. It is clear that we have $U_D \subset \mathbb{K}[Y]_{\leq D}$. Now we define

$$\chi(D) = \dim_{\mathbb{K}}(\mathbb{K}[Y]_{\leq D}) - \dim_{\mathbb{K}}(U_D).$$

Again, we can solve the system directly by linearization if $\chi(D) = 1$, but more generally, we only need $\chi(D) \leq D$. We denote by D_m the minimal degree D for which this relation is satisfied.

Let ϕ be the $\mathbb{K}$-homomorphism defined as

$$\begin{array}{ccc} \phi : \mathbb{K}[Y]_{\leq D} & \longrightarrow & \mathbb{K}[\{x_{ij} \cdot w_{ik}\}]_{\leq 2D} \\ Y_{ijk} & \longmapsto & x_{ij} w_{ik} \quad . \end{array}$$

It is clear that $\phi(\mathbb{K}[Y]_{\leq D}) = \mathbb{K}[\{x_{ij} \cdot w_{ik}\}]_{\leq 2D}$ and $\phi(U_D) = \overline{U}_D$. Let V_D be the subset of $\mathbb{K}[Y]_{\leq D}$ defined as

$$V_D = \langle \prod_{l=1}^{D-2} Y_{i_l j_l k_l} \cdot \mathcal{R} \rangle. \tag{16}$$

Lemma 2. *V_D is the kernel of the homomorphism ϕ.*

Proof. In one direction, it is clear that $V_D \subseteq \ker \phi$. Now let $\mathcal{B} = \{M_i\}$ be the canonical basis of $\mathbb{K}[Y]_{\leq D}$ and r the number of distinct monomials of type $\phi(M_i)$. It is clear that each $\phi(M_i)$ is a non-null monomial of $\mathbb{K}[\{x_{ij} \cdot w_{ik}\}]_{\leq 2D}$, and thus r is the rank of ϕ. We can then choose $b = \#\mathcal{B} - r$ linearly independent polynomials of the form $M_i + M_j$ with $\phi(M_i) = \phi(M_j)$. Since $\dim(\ker \phi) = b$, it follows that these polynomials form a basis of $\ker \phi$.

Let $M_1 = \prod_l m_{1l}$, where $m_{1l} = \prod_r Y_{i_l j_r k_r}$ are monomials involving only variables (i.e. quadratic monomials in $\mathbb{K}[\{x_{ij} \cdot w_{ik}\}]$) from the same S-Box. It is clear that $M_2 = \prod_l m_{2l}$, with $\phi(m_{1l}) = \phi(m_{2l})$. So without loss of generality, we assume that $M_1 = m_1 = \prod_r Y_{ij_r k_r}$ and $M_2 = m_2 = \prod_l Y_{ij_l k_l}$.

If we write $\nu : j_r \to k_r$ and $M_1 = \prod_j Y_{ij\nu(j)}$, then there exists a permutation $\sigma \in \mathcal{S}_K$ such that $M_2 = \prod_j Y_{i\sigma(j)\nu(j)}$. Write σ as a product of transpositions

[5] In fact, by renaming monomials if necessary, we should be able to successfully solve the system if $\overline{\chi}(D) \leq D$ [8].

$\sigma = \prod_p \tau_p$, where $\tau_p = (a_p, b_p)$ with $a_p, b_p \in \{k_r\}$. Denote by t_p the product $\tau_{p-1}\tau_{p-2}\ldots\tau_0$, where $t_0 = \mathrm{id}$ and $t_\infty = \sigma$. If we call

$$Z_{jk} = \begin{cases} Y_{ijk} & \text{if } j \neq k \\ [Y_{ijk}] & \text{if } j = k \end{cases},$$

then we have

$$\prod_i Z_{i\nu(i)} + \prod_i Z_{\tau_p(i)\nu(i)} = \left(Z_{a_p\nu(a_p)}Z_{b_p\nu(b_p)} + Z_{a_p\nu(b_p)}Z_{b_p\nu(a_p)}\right) \prod_{i\neq a_p,b_p} Z_{i\nu(i)}$$

$$\prod_i Z_{t_p(i)\nu(i)} + \prod_i Z_{t_{p+1}(i)\nu(i)} = \left(Z_{a_p\nu(a_p)}Z_{b_p\nu(b_p)} + Z_{a_p\nu(b_p)}Z_{b_p\nu(a_p)}\right) \prod_{t_p(i)\neq a_p,b_p} Z_{t_p(i)\nu(i)}.$$

Therefore

$$M_1 + M_2 = \prod_i Z_{t_\infty(i)\nu(i)} + \prod_i Z_{t_0(i)\nu(i)} \in \langle (Z_{\alpha\beta}Z_{\gamma\delta} + Z_{\alpha\delta}Z_{\gamma\beta}) \cdot \mathbb{K}[Y]_{\leq D-2} \rangle,$$

and $\ker\phi = V_D$. □

Therefore, according to the lemma we have

$$\frac{\mathbb{K}[Y]_{\leq D}}{V_D} \cong \mathbb{K}[\{x_{ij} \cdot w_{ik}\}]_{\leq 2D} \qquad \text{and} \qquad \frac{U_D}{V_D} \cong \overline{U}_D.$$

It follows that $\chi(D) = \overline{\chi}(D)$ and $P_m = D_m$.

C An Example for which the T′ Method Fails

In Appendix B of [11] a concrete working example for the T′ method is presented. The example consisted of a system of 8 quadratic equations with 5 variables, such that $T = 16$ and $T' = 10$. By alternately applying the method with respect to the variables x_1 and x_2, a total of 15 linearly independent equations were obtained and the system could then be solved by linearization.

Below we present an example for which the T′ method does not work. Our system has 7 linearly independent quadratic equations over $\mathbb{F}_2$ with 5 variables (so we have $E = 7$, $T = 16$ and $T' = 10$). Our system has also a unique solution ($x_2 = x_3 = x_5 = 0$, $x_1 = x_4 = 1$). In our case, however, there is only one exceeding equation, i.e. $C = E - T + T' = 1$.

$$\begin{cases} x_1x_2 + x_1x_4 + x_2x_3 + x_2x_5 + x_4x_5 + x_1 + x_3 + x_4 + x_5 + 1 = 0 \\ x_1x_2 + x_1x_3 + x_2x_5 + x_3x_5 + x_4x_5 + x_4 + 1 = 0 \\ x_2x_3 + x_3x_5 + x_3x_4 + x_2 + x_3 + x_4 + x_5 + 1 = 0 \\ x_1x_5 + x_1x_3 + x_3x_4 + x_4x_5 + x_5 = 0 \\ x_1x_5 + x_1x_3 + x_2x_4 + x_2 + x_3 = 0 \\ x_1x_3 + x_2x_4 + x_3x_5 + x_1 + x_2 + x_5 + 1 = 0 \\ x_2x_5 + x_2x_3 + x_4x_5 + x_2 + x_3 + x_5 = 0 \end{cases} \tag{17}$$

The system (17) is such that for every variable x_i, we have $\mathcal{C}_i \subseteq \ker(X_i)$ and therefore $X_i(\mathcal{C}_i) = \{0\}$. So we are unable to obtain a single new equation. For example, on working with the variable x_1, we can represent the system as:

$$\begin{cases} x_2x_3 = x_1x_3 + x_1x_4 + x_1x_5 + 1 \\ x_2x_4 = x_1x_3 + x_1x_5 + x_2 + x_3 \\ x_2x_5 = x_1x_3 + x_1 + x_3 + x_4 \\ x_3x_4 = x_1x_3 + x_1x_4 + x_1 + x_2 + x_4 + 1 \\ x_3x_5 = x_1x_5 + x_1 + x_3 + x_5 + 1 \\ x_4x_5 = x_1x_4 + x_1x_5 + x_1 + x_2 + x_4 + x_5 + 1 \\ 1 = x_1x_2 + x_1x_4 + x_1 + x_2 + x_4. \end{cases} \tag{18}$$

However, when multiplying the last equation by x_1 we have

$$x_1 \cdot (1 + x_1x_2 + x_1x_4 + x_1 + x_2 + x_4) = 0.$$

The same is valid for all the remaining variables. For example, with respect to x_2:

$$\begin{cases} x_1x_3 = x_2x_5 + x_1 + x_3 + x_4 \\ x_1x_4 = x_2x_3 + x_2x_4 + x_2 + x_3 + 1 \\ x_1x_5 = x_2x_4 + x_2x_5 + x_1 + x_2 + x_4 \\ x_3x_4 = x_2x_3 + x_2x_4 + x_2x_5 \\ x_3x_5 = x_2x_4 + x_2x_5 + x_2 + x_3 + x_4 + x_5 + 1 \\ x_4x_5 = x_2x_3 + x_2x_5 + x_2 + x_3 + x_5 \\ 0 = x_1x_2 + x_2x_3 + x_2x_4 + x_1 + x_3 + x_4. \end{cases}$$

Again the same occurs:

$$x_2 \cdot (x_1x_2 + x_2x_3 + x_2x_4 + x_1 + x_3 + x_4) = 0.$$

Therefore no new equations can be generated and the T′ method fails for this system.