Cryptanalysis of White-Box Implementations
(draft version)

W. Michiels[1], P. Gorissen[1], and H.D.L. Hollmann[1]

Philips Research Laboratories, High Tech Campus 34, Eindhoven, The Netherlands,
wil.michiels@philips.com

Abstract. A white-box implementation of a block cipher is a software implementation from which it is difficult for an attacker to extract the cryptographic key. Chow et al. [4, 5] published white-box implementations for AES and DES that both have been cryptanalyzed. However, these white-box implementations are based on ideas that can easily be used to derive white-box implementations for other block ciphers as well. As the cryptanalyses published use typical properties of AES and DES, it remains an open question whether the white-box techniques proposed by Chow et al. can result in a secure white-box implementation for other ciphers than AES and DES. In this paper we identify a generic class of block ciphers for which the white-box techniques of Chow et al. do not result in a secure white-box implementation. The result can serve as a basis to design block ciphers and to develop white-box techniques that do result in secure white-box implementations.

1 Introduction

Symmetric block ciphers, such as AES and DES, are designed to secure communication against an eavesdropper that has access to the communication channel. This means that the security of symmetric block ciphers is based on attack models that can be referred to as 'black box': to derive a cryptographic key, an attacker is assumed to have at most access to the input and output of a cryptographic algorithm. An attacker is assumed to have no access to the execution of the cryptographic algorithm.

Crucial for a black-box attack model to apply is that the communicating parties are trusted. However, in many applications this condition is not satisfied. If, for instance, a content provider broadcasts encrypted data to a user, then the user may benefit from illegally distributing the key for decrypting the data to other users. As a second example we mention a virus that has installed itself on a system to extract a key that is used by the system for securing financial transactions. To protect a cryptographic key in the case of a potentially malicious host, we have to deal with the severe 'white-box attack model' in which an attacker is assumed to have full access to and full control over the implementation of a cryptographic algorithm.

White-box cryptography is the discipline that aims at solving the problem of how to implement a cryptographic algorithm in software, such that the key cannot be extracted by a white-box attack. A software implementation of a cryptographic algorithm that tries to resist a white-box attack on its key is called a white-box implementation. Chow et al. present white-box implementations for the block ciphers AES and DES [4, 5]. These white-box implementations are based on similar ideas that can easily be used to derive white-box implementations of other block ciphers as well.

White-box attacks have been published for extracting the 128-bit AES key and the 56-bit DES key from the white-box AES and DES implementations of Chow et al. [2, 6–9]. However, the attacks use typical properties of AES and DES and do not translate to white-box implementations of other block ciphers. Hence, it remains an open question whether the white-box techniques proposed by Chow et al. can result in a secure white-box implementation for other ciphers than AES and DES, such as, for instance, Serpent [1]. In this paper we prove for a generic class of block ciphers that the white-box techniques of Chow et al. do not result in a secure white-box implementation. More specifically, we prove that under some mild conditions on the diffusion matrix, we can extract the round keys from the white-box implementation of an arbitrary substitution-linear transformation cipher (see Definition 1). Note that if the key scheduling algorithm is invertible, then having these round keys suffices to also derive the main key used.

Definition 1 (Substitution-Linear Transformation Cipher (SLT cipher)). *A cipher is called an SLT cipher if it can be specified as follows. It consists of R rounds. A single round r is a bijective function $\mathcal{F}^r_{\mathrm{SLT}}(x_1, x_2, \ldots, x_s)$ on $GF(2)^n$ with $x_i \in GF(2)^m$ and $n = m \cdot s$. A round consists of the following operations. It starts with adding an n-bit round key $k^r = (k^r_1, k^r_2, \ldots, k^r_s)$ to its input. That is, the value $y_i = k^r_i \oplus x_i$ is computed. Next, the round computes $z_i = S^r_i(y_i)$ for all y_i, where the (non-linear) S-boxes $S^r_1, S^r_2, \ldots, S^r_s$ are part of the cipher specification and thus key-independent. These two steps realize confusion. The diffusion is realized by multiplying the outcome $z = (z_1, z_2, \ldots, z_s)$ of the S-boxes with an $n \times n$ invertible matrix M^r. This matrix is also part of the cipher specification.*

In our notation we omit the index r denoting the round when this value is clear from the context. The remainder of this paper is organized as follows. In Section 2 we give a precise formulation of the result that is proved in this paper. Key in this formulation is a specification of the information that is available to an attacker in a white-box implementation. In Sections 3-6 we next present our cryptanalysis. To illustrate the cryptanalysis we conclude in Section 7 with showing how it can be used to extract the round keys from a white-box AES and a white-box Serpent implementation.

2 Problem formulation and notation

In order to attack the white-box implementation of an SLT cipher, we have to specify what kind of information we can obtain from it by a white-box attack.

To answer this question, we briefly discuss how Chow et al. derive a white-box implementation of a block cipher.

First, they derive for each round of the block cipher an implementation that computes its output by only performing a sequence of table lookups. The input to a lookup table is either the input to the round or it is obtained by sequencing the outputs of one ore more other lookup table. Such an implementation can be modeled by a network of lookup tables, where an arc from table T to T' means that (part of) the output of table T is used as (part of the) input to table T'.

To arrive at a white-box implementation, we next obfuscate the lookup tables by encoding their input and output. Encoding the input and output of a table T with bijective functions f_{in} and f_{out}, respectively, corresponds to replacing table T by $f_{out} \circ T \circ f_{in}^{-1}$. Hence, we incorporate in it an input *de*coding and an output *en*coding. To see that the application of encodings realizes obfuscation, observe that encoding the input of a lookup table changes the order of its rows and that encoding the output changes the value of the rows.

The lookup tables are encoded in such a way that the functionality of the entire implementation does not change. A possible strategy to do this is as follows. The first tables in the network do not get an input encoding and the last ones do not get an output encoding. Furthermore, we choose the input encoding of a table, such that it matches the encoding that has been put on its input data by the tables that directly precede it in the network. To illustrate this, suppose that the input of a table T' is exactly given by the output of a table T. We then encode the output of T by a randomly chosen encoding f. The input encoding of T' is chosen correspondingly, which means that the input decoding incorporated in T' consists of applying f^{-1}. The result is that the output encodings of T and the input decoding of T' have no net effect.

Let $(x_1, x_2, \dots, x_s)$ be the input to a round of an SLT cipher, where x_i is the m-bit input to S-box S_i. Then, before applying the encodings, the network of lookup tables has the the property that each word x_i is input to some lookup table T_i. For details, we refer to [4, 5]. For the white-box implementation obtained after applying the encodings, this means that an attacker who can observe the inputs of all tables in the implementation, which we assume to be true in a white-box attack, has access to the encoded version $f_i(x_i)$ of each value x_i. Here f_i is a secret bijective function that is used as input encoding for T_i. This brings us at the following property, which specifies the information that is available to an attacker who tries to extract the round keys from a white-box implementation based on the techniques of Chow et al.

Property 1. In a white-box attack of a white-box implementation of an SLT cipher, an attacker has for each m-bit input word x_i of round r access to the encoded version $f_i^r(x_i)$ of x_i. Function f_i^r is an arbitrary m-bit bijective function that is unknown to the attacker.

In order to formulate the main result of this paper, we need the following definitions.

Definition 2. *Let N be an $n \times n$ matrix with $n = m \cdot s$. We partition N into s vertical strips of size $n \times m$ and denote strip j with $1 \leq j \leq s$ by N_j. To be specific,*

$$(N_j)_{x,y} = N_{x,(j-1)\cdot m+y},$$

where the rows and columns are indexed with $\{1, 2, \ldots, n\}$.

We can partition N further by splitting N_j into s blocks $N_{i,j}$ of size $m \times m$ with $1 \leq i, j \leq s$. That is,

$$(N_{i,j})_{x,y} = N_{(i-1)m+x,(j-1)m+y},$$

which implies that

$$N = \begin{pmatrix} N_{1,1} & N_{1,2} & \ldots & N_{1,s} \\ N_{2,1} & N_{2,2} & \ldots & N_{2,s} \\ \vdots & \vdots & \vdots & \vdots \\ N_{s,1} & N_{s,2} & \ldots & N_{s,s} \end{pmatrix}.$$

We refer to row

$$(N_{i,1} \; N_{i,2} \; \ldots \; N_{i,s})$$

of blocks as the ith block row of N.

Definition 3. *Let N be an $n \times n$ matrix with $n = m \cdot s$. We say that a subset $U \subseteq \{1, 2, \ldots, s\}$ represents a spanning block set for block row i if the collection of all the m-bit columns from the blocks $N_{i,j}$ with $j \in U$ spans $GF(2)^m$.*

If there are two subsets $U, V \subseteq \{1, 2, \ldots, s\}$ with $U \cap V = \emptyset$ that both represent spanning block sets for block row i, then we say that block row i has disjoint spanning block sets.

The main result of this paper can now be stated as follows.

Theorem 1. *Consider an SLT cipher for which the diffusion matrices have the property that all their block rows have disjoint spanning block sets. Furthermore, consider a white-box implementation for this cipher that satisfies Property 1. We can then extract the round key of any round r with $1 < r < R$.*

The theorem above does not cover the first and last round of the cipher. The reason for this is as follows. In order to not change the functionality of the white-box implementation, the input of the first round and the output of the last round may not be encoded. However, by omitting these encodings the white-box implementation of the first and last round become less secure. As as solution to this problem, Chow et al. propose to add the encodings and to either undo these encodings elsewhere in the software or to include these encodings in the definition of the block cipher that is implemented. In both cases it will not only be the goal of an attacker to derive the round keys of the first and last round, but also to derive the encodings applied. To simplify the discussion we exclude the attack of these rounds in this paper. We note, however, that these rounds can also be attacked. The attack is based on the following result. By applying our cryptanalysis, an attacker can derive the output encoding of the first round and the input encoding of the last round. This gives the attacker the plain output of the first round and the plain input to the last round. Using this, the first and last round can be attacked.

We end this section with the description of some notational conventions used throughout this paper.

- The multiplication of a matrix N with a vector x will be written as $N \cdot x$.
- If N is a matrix, then $\mathcal{N}$ denotes the function corresponding to a matrix multiplication with N, i.e., $\mathcal{N}(x) = N \cdot x$.
- If T denotes a lookup table, then we also write T for the function defined by it.
- We define $\oplus_c$ as the function $\oplus_c(x) = x \oplus c$. Using this, we can write the key addition of a SLT cipher as $\oplus_k$.
- If we consider an affine function a, then we assume that is given by $a(x) = A \cdot x \oplus \tilde{a}$ without explicitly defining matrix A and constant $\tilde{a}$.
- Let $g_1, g_2, \ldots, g_s$ be functions on m bits. Furthermore, let $x = (x_1, x_2, \ldots, x_s) \in GF(2)^n$ with $x_i \in GF(2)^m$. Then we write

  $$g_{\text{casc}}(x) = (g_1, g_2, \ldots, g_s)(x) = (g_1(x_1), g_2(x_2), \ldots, g_s(x_s))$$

  and we say that g_{casc} is an m-bit cascade.

3 Removing non-linear part of encodings

According to Property 1, an attacker has for each input word x_i access to its encoded version $f_i(x_i)$, where f_i is an unknown bijective function. In the first step of our cryptanalysis we remove the non-linear part of this encoding.

Consider a round r with $1 \leq r < R$. Let $U = \{u_1, u_2, \ldots, u_l\}$ and $V = \{v_1, v_2, \ldots, v_{l'}\}$ be two disjoint spanning block sets for block row i of the diffusion matrix M of round r. Without loss of generality we assume that $U \cup V = \{1, 2, \ldots, s\}$, i.e., $l' = m - l$. This partitions the s input words of a round input into two parts: words that are input to an S-box S_i with $i \in U$ and words that are input to an S-box S_i with $i \in V$. We write x_i for an input word that is input to S-box S_i with $i \in U$ and we write y_i for an input word that is input to S-box S_i with $i \in V$. The vector x contains all l input words x_i and vector y contains all l' input words y_i. We then have that the ith output word z_i of round r is given by $\alpha(x, y)$ defined by

$$\alpha(x, y) = f_i^{r+1}(\beta(x) \oplus \gamma(y)),$$

where $\beta(x) = \bigoplus_{j \in U} \beta_j(x_j)$ with

$$\beta_j(x_j) = \mathcal{M}_{i,j}^r \circ S_j \circ \oplus_{k_j^r} \circ (f_j^r)^{-1}(x_j)$$

and $\gamma(y) = \bigoplus_{j \in V} \gamma_j(y_j)$ with

$$\gamma_j(y_j) = \mathcal{M}_{i,j}^r \circ S_j \circ \oplus_{k_j^r} \circ (f_j^r)^{-1}(y_j).$$

Because U and V are spanning blocks, we have that $\beta(x)$ and $\gamma(y)$ are surjective on the vector space $GF(2)^m$.

In the full version of this paper, we prove the following result.

Theorem 2. *In $\mathcal{O}(\min(s,m)\cdot 2^{3m})$ time we can construct sets W_x and W_y with $|W_x| = |W_y| = 2^m$, such that $\alpha(x,y)$ is (i) a bijection on $x \in W_x$ for any fixed y and (ii) a bijection on $y \in W_y$ for any fixed x.*

Let $\alpha_c(x) = \alpha(x,c)$. We then get that for any $c_1, c_2 \in W_y$ and $c = \gamma(c_1) \oplus \gamma(c_2) \in GF(2)^m$ we have

$$\alpha_{c_1} \circ \alpha_{c_2}^{-1}(z_i) = f_i^{r+1}(c \oplus (f_i^{r+1})^{-1}(z_i)).$$

By keeping c_1 constant and by letting c_2 cycle through all its 2^m possible values from W_y, we can realize any value $c \in GF(2)^m$. That is, we can construct a collection of 2^m lookup tables, each specifying for a different value c the function $f_i^{r+1} \circ \oplus_c \circ (f_i^{r+1})^{-1}$. We can now use the following result of Billet et al. [2] to derive the non-linear part of f_i^{r+1}.

Theorem 3. *Let a set of functions $\{f \circ \oplus_c \circ f^{-1} \mid c \in GF(2^m)\}$ be given as lookup tables, where f is an arbitrary permutation of $GF(2^m)$ and $\oplus_c$ is the translation by c in $GF(2^m)$. One can construct a function g such that there exists an affine mapping a satisfying $g = f \circ a$.*

Combining Property 1 with the theorem above yields that an attacker can get access to the following information when extracting the round keys from a white-box implementation.

Property 2. In a white-box attack of a white-box implementation of an SLT cipher, an attacker has for each m-bit input word x_i of round r with $2 \leq r \leq R$ access to the encoded version $a_i^r(x_i)$ of x_i. Here, $a_i^r(x_i) = A_i^r \cdot x \oplus \tilde{a}_i^r$ is an m-bit affine function.

4 Transformation into table network

From Property 2 it follows that after performing the first step of our cryptanalysis, an attacker has for each round r with $1 < r < R$ access to the input-output behavior of the function

$$\mathcal{G}_{\mathrm{SLT}}^r = a_{\mathrm{casc}}^{r+1} \circ \mathcal{F}_{\mathrm{SLT}}^r \circ (a_{\mathrm{casc}}^r)^{-1}. \tag{1}$$

We now present an implementation of $\mathcal{G}_{\mathrm{SLT}}$ that only consists of s lookup tables. More specifically, we define tables $T_1, T_2, \ldots, T_s$ such that $\mathcal{G}_{\mathrm{SLT}}(x_1, x_2, \ldots, x_s)$ equals $\bigoplus_{i=1}^{s} T_i(x_i)$.

Let $\bar{x_i}$ with $x_i \in GF(2)^m$ be the s-dimensional vector over $GF(2)^m$ that contains value x_i at position i and value 0 at any other position. We now define T_i as

$$T_i(x_i) = \begin{cases} \mathcal{G}_{\mathrm{SLT}}(\bar{x_1}) & \text{if } i = 1, \\ \mathcal{G}_{\mathrm{SLT}}(\bar{x_i}) \oplus \mathcal{G}_{\mathrm{SLT}}(0) & \text{otherwise.} \end{cases}$$

By the definitions of $\mathcal{G}_{\mathrm{SLT}}$ and $\mathcal{F}_{\mathrm{SLT}}$, we have

$$\mathcal{G}_{\mathrm{SLT}}(x_1, x_2, \ldots, x_s) = \bigoplus_{i=1}^{s} a_{\mathrm{casc}}^{r+1} \circ \mathcal{M}_i \circ S_i \circ \oplus_{k_i^r} \circ (a_i^r)^{-1}(x_i). \tag{2}$$

This implies that for $i = 1$ value $T_i(x_i)$ is given by

$$a_{\mathrm{casc}}^{r+1} \circ \mathcal{M}_1 \circ S_1 \circ \oplus_{k_1^r} \circ (a_1^r)^{-1}(x_1) \oplus \bigoplus_{l=2}^{s} a_{\mathrm{casc}}^{r+1} \circ \mathcal{M}_l \circ S_l \circ \oplus_{k_l^r} \circ (a_l^r)^{-1}(0) \tag{3}$$

while for $i \geq 2$ value $T_i(x_i)$ is given by

$$a_{\mathrm{casc}}^{r+1} \circ \mathcal{M}_i \circ S_i \circ \oplus_{k_i^r} \circ (a_i^r)^{-1}(x_i) \oplus a_{\mathrm{casc}}^{r+1} \circ \mathcal{M}_i \circ S_i \circ \oplus_{k_i^r} \circ (a_i^r)^{-1}(0). \tag{4}$$

From (2), (3), and (4) it easily follows that $\mathcal{G}_{\mathrm{SLT}}(x_1, x_2, \ldots, x_s)$ is given by $\bigoplus_{i=1}^{s} T_i(x_i)$, which was our goal.

5 Transformation into SAT cipher

In an SLT cipher, we can merge the key-addition operation into the S-box. The resulting S-box is then given by $S_i \circ \oplus_{k_i}$. Hence, an SLT cipher can be viewed as a generic SAT cipher as defined below.

Definition 4 (Generic Substitution-Affine Transformation Cipher (generic SAT cipher)). *A cipher is called a generic SAT cipher if it can be specified as follows. It consists of R rounds. A single round r is a bijective function $\mathcal{F}_{\mathrm{gen-SAT}}(x_1, x_2, \ldots, x_s)$ on $GF(2)^n$ with $x_i \in GF(2)^m$ and $n = m \cdot s$. A round consists of the following operations. First, the values $y_i = Q_i^r(x_i)$ are computed for all input words x_i, where the specification of the S-boxes Q_i^r are derived from the key. Next, an invertible affine function $b^r(y) = B^r \cdot y \oplus \tilde{b}^r$ is applied to the outcome $y = (y_1, y_2, \ldots, y_s)$ of the S-boxes. Also the specification of this affine function b^r is derived from the key.*

By (1) the round function $\mathcal{G}_{\mathrm{SLT}}^r$ is obtained from $\mathcal{F}_{\mathrm{SLT}}^r$ by encoding each m-bit input to an S-box S_i^r by an affine function a_i^r and by encoding each m-bit output word of the round by an affine function a_i^{r+1}. Hence, besides $\mathcal{F}_{\mathrm{SLT}}^r$ also round function $\mathcal{G}_{\mathrm{SLT}}^r$ can be specified as a round of a generic SAT cipher. In this section we show how to derive the round key for which a round of the generic SAT cipher corresponds to $\mathcal{G}_{\mathrm{SLT}}^r$, i.e., to the function implemented by the lookup table network $T_1, T_2, \ldots, T_s$ derived in the previous section.

From (3) and (4) it follows that T_i is given by

$$\mathcal{T}_i(x) = \mathcal{C}_i \circ S_i \circ \oplus_{k_i} \circ (a_i^r)^{-1}(x) \oplus \tilde{c}_i, \tag{5}$$

for a constant $\tilde{c}_i$ and $n \times m$ matrix C_i defined by

$$C_i = \begin{pmatrix} A_1^{r+1} & 0 & 0 \dots & 0 \\ 0 & A_2^{r+1} & 0 \dots & 0 \\ \vdots & \vdots & \vdots \; \vdots & \vdots \\ 0 & 0 & 0 \dots & A_s^{r+1} \end{pmatrix} \cdot M_i.$$

Because A_i^{r+1} and M are invertible matrices, also $n \times n$-bit matrix $(C_1, C_2, \dots, C_s)$ is an invertible matrix. This implies that the m columns of $n \times m$ matrix C_i are all independent. Let U_i be the m-dimensional vector space spanned by these independent columns. Then it follows from (5) that the 2^m rows of table T_i are exactly all the 2^m points in the space $U_i \oplus \tilde{c}_i$, i.e., in the space obtained by translating vector space U_i over $\tilde{c}_i$.

After these observations, we are ready to specify the round key of the generic SAT cipher corresponding to $\mathcal{G}_{\mathrm{SLT}}$. From each table T_i we select an arbitrary n-bit row v_i, and we define the constant value $\tilde{b}$ in round r of the generic SAT cipher as

$$\tilde{b} = \bigoplus_{i=1}^{s} v_i.$$

If we add v_i to all rows of a table T_i, then its 2^m rows are exactly all the points in the m-dimensional vector space U_i. This can be seen as follows. The rows of T_i form the space $U_i \oplus \tilde{c}_i$. Hence, $v_i = u \oplus \tilde{c}_i$ for an $u \in U_i$. By adding v_i to all rows, the space $U_i \oplus \tilde{c}_i$ defined by the rows of T_i is translated to U_i because

$$U_i \oplus \tilde{c}_i \oplus v_i = U_i \oplus \tilde{c}_i \oplus u \oplus \tilde{c}_i = U_i \oplus u = U_i.$$

We derive m independent vectors $u_1, u_2, \dots, u_m$ that span the vector space U_i, and we define the m columns of submatrix B_i of B as these m independent vectors. We then have that each value $T_i(x_i) \oplus v_i$ can be written in a unique way as a linear combination of the m columns of B_i, i.e.,

$$T_i(x_i) \oplus v_i = \bigoplus_{j=1}^{m} g_j(x_i) \cdot u_j$$

where $g(x_i) : 2^m \to 2^m$ is a bijective function and $g_j(x_i)$ gives the jth bit of $g(x_i)$. We define S-box Q_i as the function g. This concludes our definition of the round key of the generic SAT cipher. To see that by choosing this round key the generic SAT cipher indeed corresponds to $\mathcal{G}_{\mathrm{SLT}}$, observe that

$$\begin{aligned} \mathcal{F}_{\mathrm{gen-SAT}}(x_1, x_2, \dots, x_s) &= \tilde{b} \oplus \bigoplus_{i=1}^{s} \mathcal{B}_i \circ Q_i(x_i) \\ &= \bigoplus_{i=1}^{s} v_i \oplus \bigoplus_{i=1}^{s} \bigoplus_{j=1}^{m} g_j(x) \cdot u_j \end{aligned}$$

$$= \bigoplus_{i=1}^{s} v_i \oplus \bigoplus_{i=1}^{s} (T_i(x_i) \oplus v_i)$$
$$= \mathcal{G}_{\text{SLT}}(x_1, x_2, \dots, x_s).$$

6 Extracting the key

In this chapter we describe the last step of our cryptanalysis. We adopt the following strategy. First, we derive a relation between the S-boxes S_i^r of the white-boxed SLT cipher and the S-boxes Q_i^r specified by the key of the generic SAT cipher that we derived in Section 5. This relation will be of the form $Q_i^r = c_i^r \circ S_i^r \circ d_i^r$ for affine functions c_i^r, d_i^r. The function d_i^r depends on both the round key k^r of $\mathcal{F}_{\text{SLT}}^r$ and the affine encodings a_i^r that $\mathcal{G}_{\text{SLT}}^r$ puts on the input of round $\mathcal{F}_{\text{SLT}}^r$. The function c_i^r depends on the encoding a_i^{r+1} that $\mathcal{G}_{\text{SLT}}^r$ puts on the output of $\mathcal{F}_{\text{SLT}}^r$ by $\mathcal{G}_{\text{SLT}}^r$. By comparing the functions c_i^{r-1} and d_i^r, we can find the key k^r contained in d_i^r.

We now make the last step of our cryptanalysis more precise. The outcome of the previous step are S-boxes Q_i and an affine function b, such that $\mathcal{G}_{\text{SLT}} = b \circ Q_{\text{casc}}$. By (1) an equivalent definition of $\mathcal{G}_{\text{SLT}}$ is given by

$$a_{\text{casc}}^{r+1} \circ \mathcal{M} \circ S_{\text{casc}} \circ \oplus_k \circ (a_{\text{casc}}^r)^{-1}.$$

Using this and the observation that the functions b, a_{casc}^{r+1}, $\mathcal{M}$, $\oplus_k$, and a_{casc}^r are all affine, we get

$$Q_{\text{casc}} = c_{\text{casc}} \circ S_{\text{casc}} \circ d_{\text{casc}} \tag{6}$$

for affine functions

$$c_{\text{casc}} = b^{-1} \circ a_{\text{casc}}^{r+1} \circ \mathcal{M} \tag{7}$$

and

$$d_{\text{casc}} = \oplus_k \circ (a_{\text{casc}}^r)^{-1}. \tag{8}$$

In the notation we indicated that c and d are cascades. For d this is obviously true. For c this property follows from (6) and the observation that d_{casc}, Q_{casc}, and S_{casc} are cascades. Remark that (6) is equivalent to the condition that $Q_i = c_i \circ S_i \circ d_i$ for all i. Biryukov et al. [3] present an algorithm for efficiently deriving the set γ_i of all pairs (c_i, d_i) of affine functions that satisfy this equation. We define Γ such that $(c_{\text{casc}}, d_{\text{casc}}) \in \Gamma$ if $(c_i, d_i) \in \gamma_i$ for all i. Note that this set Γ contains the pair $(c_{\text{casc}}, d_{\text{casc}})$ satisfying (6), (7), and (8). To complete our cryptanalysis it now suffices to solve the following two problems.

- Which pairs $(c_{\text{casc}}^{r-1}, d_{\text{casc}}^{r-1}) \in \Gamma^{r-1}$ and $(c_{\text{casc}}^r, d_{\text{casc}}^r) \in \Gamma^r$ of affine functions satisfy (7) and (8)?
- If we have the pairs $(c_{\text{casc}}^{r-1}, d_{\text{casc}}^{r-1}) \in \Gamma^{r-1}$ and $(c_{\text{casc}}^r, d_{\text{casc}}^r) \in \Gamma^r$ of affine functions that satisfy (7) and (8), how can we derive round key k^r from this?

Observe that the former problem need not be solved completely. If we can limit the number of candidate pairs to a value l, then we can apply an algorithm for the second problem to all l candidate solutions to obtain l candidate round keys. The correct round key can next be derived by exhaustive search.

From (7) for round $r-1$ and (8) for round r it follows that $b^{r-1} \circ c^{r-1}_{\text{casc}} = a^{r}_{\text{casc}} \circ \mathcal{M}^r$ and $a^r_{\text{casc}} = (d^r_{\text{casc}})^{-1} \circ \oplus_{k^r}$, respectively. If we write $c^{r-1}_{\text{casc}}(x) = C^{r-1} \cdot x \oplus \tilde{c}^{r-1}$ and $d^r_{\text{casc}}(x) = D^r \cdot x \oplus \tilde{d}^r$ for matrices C^{r-1}, D^r and constants $\tilde{c}^{r-1}, \tilde{d}^r$, then combining these equations yields

$$\mathcal{B}^{r-1} \circ \mathcal{C}^{r-1}(x) \oplus \tilde{b}^{r-1} \oplus \mathcal{B}^{r-1}(\tilde{c}^{r-1}) = (\mathcal{D}^r)^{-1} \circ \mathcal{M}^{r-1}(x) \oplus (\mathcal{D}^r)^{-1}(k^r \oplus \tilde{d}^r).$$

The linear parts as well as the constant parts of both sides of the equation have to be equal. The equality of the linear parts implies that in order to satisfy (7) and (8), pairs $(c^{r-1}_{\text{casc}}, d^{r-1}_{\text{casc}}) \in \Gamma^{r-1}$ and $(c^r_{\text{casc}}, d^r_{\text{casc}}) \in \Gamma^r$ of affine functions have to satisfy

$$B^{r-1} \cdot C^{r-1} = (D^r)^{-1} \cdot M^{r-1}.$$

Furthermore, the equality of the constant parts implies that if the pairs satisfy (7) and (8), then the round key k^r is given by

$$k^r = D^r \cdot B^{r-1} \cdot \tilde{c}^{r-1} \oplus D^r \cdot \tilde{b}^{r-1} \oplus \tilde{d}^r.$$

We now arrive at the algorithm described in Figure 1 for finding k^r. For implementing Step 1 in the algorithm we already referred to [3]. We now describe how Step 2 can be implemented.

Known: Q_{casc}, B, S_{casc}, M.

- Step 1: Derive for a particular pair $(r-1, r)$ of successive rounds for each S-box S_i the set
$$\gamma_i = \{(c_i, d_i) \mid Q_i = c_i \circ S_i \circ d_i \wedge c, d \text{ affine}\}$$
and let Γ be such that $(c_{\text{casc}}, d_{\text{casc}}) \in \Gamma$ if $(c_i, d_i) \in \gamma_i$ for all i.
- Step 2: Derive the subset $V^r \subseteq \Gamma^{r-1} \times \Gamma^r$ of affine functions $(c^{r-1}_{\text{casc}}, d^{r-1}_{\text{casc}}) \in \Gamma^{r-1}$ and $(c^r_{\text{casc}}, d^r_{\text{casc}}) \in \Gamma^r$ such that
$$B^{r-1} \cdot C^{r-1} = (D^r)^{-1} \cdot M^{r-1}, \tag{9}$$
where matrices C^{r-1} and D^r define the linear part of c^{r-1}_{casc} and d^r_{casc}, respectively.
- Step 3: The round key k^r of round r is contained in the set
$$K^r = \left\{D^r \cdot B^{r-1} \cdot \tilde{c}^{r-1} \oplus D^r \cdot \tilde{b}^{r-1} \oplus \tilde{d}^r \mid (c^{r-1}_{\text{casc}}, d^{r-1}_{\text{casc}}, c^r_{\text{casc}}, d^r_{\text{casc}}) \in V^r\right\}.$$

Fig. 1. Basic algorithm for finding the round key of a round r

6.1 Solving the linear equivalence problem for matrices

Step 2 of the algorithm of Figure 1 deals with the matrices C and D specifying the linear parts of the affine m-bit cascade functions c_{casc} and d_{casc}. To describe the format of C and D, let $c_i = C_i \cdot x \oplus \tilde{c}_i$ and $d_i = D_i \cdot x \oplus \tilde{d}_i$ for $m \times m$ matrices C_i, D_i and m-bit constants $\tilde{c}_i$, $\tilde{d}_i$. Then it can be verified C and D are block diagonal matrices, as defined below, where the ith diagonal block is given by C_i and D_i, respectively.

Definition 5. *Let G be an $n \times n$ matrix with $n = m \cdot s$. We partition G into s^2 blocks $G_{i,j}$ of size $m \times m$ as described in Definition 2. Matrix G is called a block diagonal matrix if all off-diagonal blocks of G are zero matrices. Furthermore, we denote the ith diagonal block by G_i.*

Let $U = U_1 \times U_2 \times \ldots \times U_s$, where U_i contains $m \times m$ matrices. We say that $G \in U$ if G is a block diagonal matrix with $G_i \in U_i$ for all i.

We can now formulate the problem of Step 2 as an instance of the Linear Equivalence Problem of Matrices (LEPM); see Definition below. In the remainder of this section we discuss in a self-contained way how this problem can be solved. For the sake of readability, we can therefore reuse variables that already have an interpretation elsewhere in this paper.

Definition 6 (Linear Equivalence Problem of Matrices (LEPM)). *A problem instance is defined by (M, N, U, V) for $n \times n$ matrices M and N and sets $U = U_1 \times U_2 \times \ldots \times U_s$ and $V = V_1 \times V_2 \times \ldots \times V_s$, where U_i and V_j contain invertible $m \times m$ matrices and $n = m \cdot s$. Find all pairs of $n \times n$ matrices $(A, B) \in U \times V$ such that $N = A \cdot M \cdot B$.*

In Figure 2 we describe an algorithm for solving LEPM. The algorithm starts with reducing the sets U_i and V_j. This is done as follows. Suppose that the pair $A, B \in U \times V$ satisfies $N = A \cdot M \cdot B$. Then this means that for all pairs i, j we have $N_{i,j} = A_i \cdot M_{i,j} \cdot B_j$. Hence, if for a particular $A'_i \in U_i$ no $B'_j \in V_j$ exists with $N_{i,j} = A'_i \cdot M_{i,j} \cdot B'_j$, then we know that we can safely remove A'_i from U_i to solve LEPM. Similarly, if for an $B'_j \in V_j$ no $A'_i \in U_i$ exists with $N_{i,j} = A'_i \cdot M_{i,j} \cdot B'_j$, then we can safely remove B'_j from V_j.

If, after this reduction step, a set U_i or V_j exists that is empty, then we know that the LEPM problem instance has no solutions. Furthermore, it can be showed that if one empty set exists, then all sets are empty. Hence, it suffices to check for one set whether it is empty.

Next, suppose that all sets U_i and V_j contain exactly one linear mapping. Then the only candidate solution to the LEPM problem instance is the solution defined by these linear mappings. It follows from the reduction step that all linear mappings A_i, B_j satisfy $N_{i,j} = A_i \cdot M_{i,j} \cdot B_j$. Hence, the candidate solution is also a valid solution. This solves the LEPM problem for this case.

The only case that remains is that either a set U_i or a set V_j exists that contains more than one linear mapping. It can be proved that if no set V_j exists that contains more than one linear mapping, then neither does a set U_j exist that contains more than one linear mapping. Hence, we only have to consider the case that a set V_j exists that contains more than one linear mapping. We define an equivalence relation $\sim_j$ on $U \times V$ such that $(A, B) \sim_j (A', B')$ if and only if $B_j = B'_j$. The equivalence relation $\sim_j$ partitions the pairs (A, B) of matrices that satisfy $N = A \cdot M \cdot B$ into classes. For each matrix $B_j \in V_j$, the algorithm derives all pairs (A, B) in the corresponding class by solving the problem instance obtained by setting V_j to the singleton set $\{B_j\}$. The solution to the LEPM problem is then given by the union of all classes.

```
algorithm LEPM_solver(U, V)
begin
    repeat
        for all U_i do
            for all A_i ∈ U_i do
                if ¬∃_{B_j ∈ V_j} N_{i,j} = A_i · M_{i,j} · B_j then
                    U_i := U_i \ {A_i};
        for all V_j do
            for all B_j ∈ V_j do
                if ¬∃_{A_i ∈ U_i} N_{i,j} = A_i · M_{i,j} · B_j then
                    V_j := V_j \ {B_j};
    until U and V do not change;
    if U_1 = ∅ then
        return ∅;
    else if ∀_i|U_i| = 1 ∧ ∀_j|V_j| = 1 then
        return {(A, B)} with A_i ∈ U_i and B_j ∈ V_j;
    else /* case ∃_i|V_j| > 1 */
        select smallest j with |V_j| > 1;
        return ⋃_{B_j ∈ V_j} LEPM_solver(U, V(V_j = {B_j}));
end;
```

Fig. 2. Algorithm for solving LEPM problem in pseudo code. In the algorithm $V(V_j = \{B_j\})$ denotes V, where V_j is replaced by $\{B_j\}$.

To know whether the algorithm presented is effective for attacking a white-box implementation, we have to know an upper bound on the number of solutions returned and the number of recursive invocations. The former number is related to the cardinality of the set K of candidate round keys in the algorithm of Figure 1. The latter number determines the time complexity of the algorithm of Figure 2. The problem is that we do not want to answer the question for one particular white-box implementation of a block cipher, but for any white-box implementation of that block cipher. Hence, we want to derive upper bounds on these numbers that only depend on the block cipher specification and not, for instance, on the encodings put on the input and output of a round $\mathcal{F}_{\mathrm{SLT}}$ by the white-box implementation. The following theorem, which is proved in the full paper, can be used to derive such bounds.

Theorem 4. *Let for a round r of an SLT cipher $I = (N, M, U, V)$ be the problem instance of LEPM that is associated with the cryptanalysis of its white-box implementation. Furthermore, let $I' = (M, M, U', V')$ be the problem instance in which U_i' is given by the set of all invertible $m \times m$ matrices and V_i' by*

$$V_i = \{G_i \mid S_i^r = g_i \circ S_i^r \circ h_i \wedge g_i, h_i \text{ affine}\}$$

Then, compared to I', problem instance I does neither result in more recursive invocations nor in more solutions when applying the algorithm of Figure 2.

7 Proof of concept

As proof of concept, we indicate that our cryptanalysis can be used to attack white-box AES and white-box Serpent. It can be verified that the diffusion matrices of both AES and Serpent satisfy the property that all their block rows have disjoint spanning blocks. Recall that this is a necessary property to perform the first step of the cryptanalysis. After applying the steps described in Sections 3-5, the cryptanalysis runs the algorithm of Figure 1 to find a set K^r of candidate round keys for a given round r. The algorithm first derives for each S-box S_i the set γ_i. For the AES S-boxes these sets can be shown to have a cardinality of 2040, while for the Serpent S-boxes the cardinality is either 4 or 1. Next, the algorithm solves an LEPM problem instance to find the set V^r. As each affine function c_i and d_i from a pair $(c_i, d_i) \in \gamma_i$ can be shown to have a unique linear part for AES and Serpent, the size of set V^r is given by the number of solutions of this LEPM problem instance. Using Theorem 4 it can be proved that for any LEPM problem instance associated with a white-box implementation the algorithm does not go into recursion and that it returns only one solution. As a consequence, V^r consists of one solution. It now follows from the third step of the algorithm of Figure 1 that the set K^r of candidate round keys consists of only one solution as well, which must thus be the round key we are looking for.

References

1. Anderson, R.J., Biham, E., Knudsen, L.R.: Serpent: A proposal for the advanced encryption standard. Proceedings of the First AES Candidate Conference, 1998.
2. Billet, O., Gilbert, H., Ech-Chatbi, C.: Cryptanalysis of a White-Box AES Implementation. Proceedings of the 11th Annual Workshop on Selected Areas in Cryptography, 227–240, 2004.
3. Biryukov, A., De Cannire, C., Braeken, A., Preneel, B.: A Toolbox for Cryptanalysis: Linear and Affine Equivalence Algorithms. Proceedings of Eurocrypt 2003, 33–50, 2003.
4. Chow, S., Eisen, P., Johnson, H., van Oorschot, P.C.: A White-Box DES Implementation for DRM Applications. Proceedings of the 2nd ACM Workshop on Digital Rights Management, 1–15, 2002.

5. Chow, S., Eisen, P., Johnson, H., van Oorschot, P.C.: White-Box Cryptography and an AES Implementation. Proceedings of the 9th Annual Workshop on Selected Areas in Cryptography, 250–270, 2002.
6. Goubin, L., Masereel, J.M., Quisquater, M.: Cryptanalysis of White-Box DES Implementations. Proceedings of the 14th Annual Workshop on Selected Areas in Cryptography, 278–295, 2007.
7. Jacob, M., Boneh, D., Felten, E.: Attacking an Obfuscated Cipher by Injecting Faults. Proceedings of the Digital Rights Management Workshop, 16–31, 2002.
8. Link, H.E., Neumann, W.D.: Clarifying Obfuscation: Improving the Security of White-Box DES. International Symposium on Information Technology: Coding and Computing, 679–684, 2005.
9. Wyseur, B., Michiels, W., Gorissen, P., Preneel, B.: Cryptanalysis of White-Box DES Implementations with Arbitrary External Encodings. Proceedings of the 14th Annual Workshop on Selected Areas in Cryptography, 264–277, 2007.