Advances in automatic cryptanalysis of block ciphers

Candidate:  
Stefania Todisco  
279614

Supervisor:  
Prof. Marco Pedicini

Academic Year: 2012/2013

MSC AMS: 94A60; 68P25.
KEYWORDS: Computational Security, Exhaustive Search, Differential Cryptanalysis, Linear Cryptanalysis, Bicliques Attacks.
Synthesis

Nowadays the internet occupies large part of our lives, from emails to bank accounts: this means that being able to communicate over secure channels is of great importance.

Let suppose there are two users, Alice and Bob, who want to communicate over an insecure channel, for example the Internet. The actual problem starts with the opponent, Oscar, who has access to the channel, for instance, by hacking into an Internet router. Oscar could simply read the content of this communication, but in the worst case, he could even change it.

In order to maintain the content of the communication secret, Alice encrypts her message (called plaintext) yielding an encrypted message (called ciphertext) and sends it to Bob. Bob receives this ciphertext and decrypts the message. Decryption is thus the inverse process of encryption (Figure 0.1).

![Communication over a secure channel](image)

Formally, we can represent this idea with the concept of cryptosystem.

**Definition 0.1 (Cryptosystem)** A cryptosystem is quintuple \((P, C, K, E, D)\), where

- \(P\) is a finite set of plaintext,
- \(C\) is a finite set of ciphertext,
- \(K\) is the key space and
- for each \(k \in K\) there exists an encryption function \(e_k \in E\) and a corresponding decryption function \(d_k \in D\). Each \(e_k : P \to C\) and \(d_k : C \to P\) are functions such that \(d_k(e_k(x)) = x\) for every plaintext element \(x \in P\).
Cryptographic algorithms can be divided into: symmetric (or public-key) algorithms and asymmetric (or private-key) algorithms.

Public-key algorithms are algorithms in which encryption key can be calculated from the decryption key and viceversa. They require sender and receiver to agree on a key before they can communicate in a secure way. As long as the communication needs to remain secret, the key must be kept secret. Private-key algorithms, instead, are algorithms designed to use different keys for encryption and decryption processes. They are called “public-key” algorithms because the encryption key can be made public; anyone can use it to encrypt a message, but only a specific person having the correct decryption key can decrypt it.

The strength of a cryptographic system is assessed by means of its resistance to several types of attacks, which can be classified as follows:

- ciphertext-only attack - the attacker is assumed to be able to access only a set of ciphertexts.
- known-plaintext attack - the attacker has samples of both the plaintext and its related ciphertext.
- chosen-plaintext attack - the attacker has the capability to choose arbitrary plaintexts to be encrypted and obtain the corresponding ciphertexts.
- chosen-ciphertext attack - the attacker has a chance to enter one or more known ciphertexts and obtain the resulting plaintexts.
- related-key attack - the attacker can observe the operation of a cipher under several different keys whose values are initially unknown, but where some mathematical relationships connecting the keys are known to the attacker.

The basic attack to a cryptosystem is given by the brute force attack, whose complexity provides the upper bound for a generic attack complexity. The idea behind this kind of attack is that, whenever we get a plaintext-ciphertext pair \((P, C)\), in order to determine the key \(K\), we try all the keys until we find one which verifies \(e_K(P) = C\). This technique requires a large number of operations, therefore, in order to determine the key, it is necessary to have either a high computing power or a lot of time. Conversely, if we know that the same plaintext \(P\) can be encrypted many times, to speed up the search we might be thinking to encrypt the plaintext \(P\) using all the possible keys \(K\) and store these results in a lookup table. At this point whenever we get a new ciphertext \(C\) related to the plaintext \(P\), we need only to check for that ciphertext in the lookup table in order to find the key. Although with this technique the key can be carried out almost instantly, on the other hand, it requires a large amount of memory.

On the base of what we said so far, we can deduce that one of cryptanalysis goals is to determine attacks with complexity lower than the exhaustive search having a good balance between the amount of memory required and time consumption.

In this thesis we focused our attention on the advances in automatic cryptanalysis of block ciphers.

Block ciphers are deterministic and symmetric key algorithms working on finite length bit groups organized in a block. Most modern-day ciphers are iterated ciphers, in which simple functions are combined together and iterated several times. Each iteration is called a round while the composition of these simple functions is defined as round function.
A plaintext is encoded through \(N_r\) rounds, therefore, in order to define an iterated cipher in its whole, the definition of a key schedule function generating the specific round key is also required. This kind of ciphers frequently incorporate permutation and substitution operations as well as modular arithmetic operations. Substitution functions are known as \(S\)-boxes and can be defined as

\[
S : \{0,1\}^n \rightarrow \{0,1\}^t
\]

where \(n\) and \(t\) are two integers greater than or equal to 1. Having \(S\)-boxes with good properties is then essential to the security of block ciphers. Concepts reported hereafter are very important to analyse these properties, in particular linear and differential vulnerability of an \(S\)-box.

**Definition 0.2 (Linear Characteristics)** Let \(S\) be an \(S\)-box with \(n\)-bit input and \(t\)-bit output. The linear characteristics of \(S\) are denoted by \(L_{M^*}(S)\) and stand for the difference between the number of \(n\)-bit values \(x\) such that the two bitwise scalar products \((x \cdot M)\) and \((m \cdot S(x))\) are equal modulo 2 and the number of pairs such that these scalar products are different, that is

\[
L_{M^*}(S) = \sum_{(x \cdot M) \equiv (m \cdot S(x)) \pmod{2}} 1 - \sum_{(x \cdot M) \not\equiv (m \cdot S(x)) \pmod{2}} 1
\]

The \(n\)-bit value \(M\) is called the input mask and the \(t\)-bit value \(m\) the output mask.

**Definition 0.3 (Differential Characteristics)** Let \(S\) be an \(S\)-box with \(n\)-bit input and \(t\)-bit output. The differential characteristics of \(S\) are denoted by \(D_{\Delta}(S)\) and stand for the number of input pairs \((x, y)\) with input difference \(\Delta\) and output difference \(\delta\), i.e., \(D_{\Delta}(S)\) is the set

\[
D_{\Delta}(S) = \{(x, x^*) | x \oplus x^* = \Delta \text{ and } S(x) \oplus S(x^*) = \delta\}
\]

**Definition 0.4 (Propagation Ratio)** The propagation ratio for the differential \((\Delta, \delta)\) is defined as follows:

\[
R_P(\Delta, \delta) = \frac{D_{\Delta}(S)}{2^n}
\]

**Definition 0.5 (n-round differential trail)** An \(n\)-round differential trail is a tuple

\[
\Delta = (\Delta^{(1)}, \delta^{(1)}, \ldots, \Delta^{(n)}, \delta^{(n)})
\]

where \(\Delta^{(i)}\) and \(\delta^{(i)}\) represent, respectively, the input and the output differential of round \(i\). Moreover \(\delta^{(i)} = \Delta^{(i+1)}\), for all \(i = 1, \ldots, n - 1\).

An \(n\)-round differential trail having \(\Delta^{(1)} = \delta^{(n)}\) is called an \(n\)-round iterative differential trail.

If we make the assumption that all the propagation ratios in an \(n\)-round differential trail are independent, then the propagation ratio for the whole differential trail is given by the product of the propagation ratios of all the differentials involved in the trail, that is

\[
R_P(\Delta, \delta) = \prod_{i=1}^{n} R_P(\Delta, \delta)
\]
We used the above defined concepts applying them to the analysis of the S-box properties of an existing cipher, namely PRESENT.

PRESENT is an ultra-lightweight block cipher proposed by A. Bogdanov et al. in CHES 2007 [BKL+07]. It is a 31-round Substitution-Permutation Network operating on 64-bits text block which uses key size of either 80 or 128 bits. The non-linear substitution layer of PRESENT is similar to the one of Serpent and uses only one 4-bits S-box $S$ which is applied 16 times in parallel in each round, while its linear permutation layer is similar to the one of DES.

We chose to analyse PRESENT because its S-box is rather simple, thus lending itself well to be analysed. We also knew it existed an attack, exploiting a 4-rounds iterative differential trail, to one of its reduced round versions [Wan07]. Therefore we decided to determine such iterative differential trail.

We started with determining the differential characteristics table for PRESENT S-box, whose elements represent differentials $D_{\Delta}(S)$, each of them giving rise to a propagation ratio $R_P(\Delta, \delta) = D_{\Delta}(S)/2^4$. In order to determine the iterative differential trail having the highest propagation ratio, we aimed at having only two active S-boxes per round, therefore we selected only those differential pairs allowing us such condition. Concatenating them, we could determine a differential trail.

Since our goal was to find a 4-rounds iterative differential trail, the idea exploited was to represent this set of differentials through a directed graph\(^1\) in which each node represents a differential and each edge, connecting two nodes $\Delta$ and $\delta$, represents the related propagation ratio $R_P(\Delta, \delta)$. In this way, the problem of finding the 4-rounds differential trail, can be led back to the problem of searching a loop on the graph having length 4 and the highest propagation ratio (calculated as the product of weights). The graph obtained is shown in Figure 0.2.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{PRESENT_Graph_representation.png}
\caption{PRESENT Graph representation}
\end{figure}

\footnote{A directed graph or digraph is an ordered pair $D = (V, E)$ with $V$ a set whose elements are called vertices or nodes, and $E$ a set of ordered pairs of vertices, called arcs, directed edges, or arrows. An arc $a = (x, y)$ is considered to be directed from $x$ to $y$; $y$ is called the child node and $x$ is called the parent node of the arc.}
Once defined this graph, we executed an additional filtering operation consisting in selecting from the graph only those nodes belonging to the *connected components*\(^2\) presenting more than one node, consequently obtaining a second graph, as it is represented in Figure 0.3.

![PRESENT Graph connected component](image)

**Fig. 0.3.** PRESENT Graph connected component

At this point, we analysed the graph looking for loops having length 4 and a weight corresponding to the product between the different propagation ratios related to each edge.

The outcome of this analysis was that it exists only one 4-rounds iterative differential trail (shown in Figure 0.4) having propagation ratio

\[
R_P(\Delta, \delta) = (2^{-4})^3 \cdot 2^{-6} = 2^{-18}. \tag{0.7}
\]

![Differential path representation](image)

**Fig. 0.4.** Differential path representation

\(^2\) In graph theory, a *connected component* of an undirected graph is a subgraph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the supergraph.
After having found the iterative differential trail, we focused our attention on computing linear and differential characteristics tables which are the starting point for linear and differential attacks. Since we know it is possible to compute these tables through the Walsh transform [Jou09], we used it to calculate those related to PRESENT.

Walsh (or Hadamard-Walsh) transform is a kind of discrete Fourier transform which has been used for a long time in coding theory. Let us suppose to have \( t > 1 \) boolean functions \( f_1, \ldots, f_t \), where for boolean function we mean a function \( f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2 \). Let also \( F : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^t \) be the function which maps every \( x \in \mathbb{F}_2^n \) with

\[
F(x) = (f_1(x), \ldots, f_t(x)).
\]

(0.8)

Functions \( f_1, \ldots, f_t \) are called the coordinate functions of \( F \) while the function \( F \) is named an \((n, t)\)-function.

**Definition 0.6 (Walsh Transform)** The Walsh Transform of a function \( F : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^t \) is defined as the function mapping every ordered pair \((M, m) \in \mathbb{F}_2^n \times \mathbb{F}_2^t\) in the value in \( M \) of the Walsh transform of the component function \( m \cdot F \), that is

\[
W(F) := (m \cdot F)_\chi = \sum_{x \in \mathbb{F}_2^n} (m \cdot F)_\chi(x)(-1)^{M \cdot x} = \sum_{x \in \mathbb{F}_2^n} (-1)^{m \cdot F(x) \oplus M \cdot x}.
\]

(0.9)

As PRESENT S-box is a function \( S \) having several output bits, in order to apply Algorithm 0.1, we defined an auxiliary table given by the following scalar product \( S_m(x) = (m \cdot S(x)) \), where \( m \) and \( x \) vary in \( \mathbb{F}_2^4 \). Therefore, we encoded function \( S \) as an array of 1 and \(-1\) in the following way

\[
f_{S_m}(x) = \{(-1)^{S(x) \cdot m}, x \in \mathbb{F}_2^4\}.
\]

(0.10)

for all \( m \in \mathbb{F}_2^4 \). Once this array was defined, for each value of \( m \), we used the pseudocode reported in Algorithm 0.1 to compute the Walsh transform obtaining the required linear characteristics table.

We then computed the differential characteristic table, considering again the table \( S_m(X) = m \cdot S(x) \), with \( x \) and \( m \) varying in \( \mathbb{F}_2^4 \). For each value of \( m \in \mathbb{F}_2^4 \), and then for each column of the table \( S_m(X) \), we computed Walsh transform, we pointwisely squared each component of the resulting array, and then we computed the inverse Walsh transform using the pseudocode reported in Algorithm 0.2.

We therefore had to apply a further inverse Walsh transform, this time reading the previously obtained table by rows rather than by columns. The table resulting from the latter operation, contains the required differential characteristics. Source codes related to the analysis just described are reported in Chapter 4 of the thesis.

After having analysed PRESENT our attention focused on the Advanced Encryption Standard. The fact that an attack to full AES having complexity less than the exhaustive search has been achieved only recently by Bogdanov et al. [BKR11] through the use of bicliques, was for us very
interests; that is why we decided to analyse such an attack. At the same time, being the AES encryption function complex - before applying our analysis directly on AES - we decided to start from a cipher having a structure similar to, but simpler than that of AES, namely KLEIN, which allowed us to analyse in detail how bicliques attacks operate.

Bicliques cryptanalysis is a rather young technique first introduced by Khovratovich et al. [KRS11] in 2011 and then presented to the FSE in 2012 [KRS12].

**Definition 0.7** A biclique is a complete bipartite graph connecting every element in a set of starting states $S$ with every element in a set of ending states $C$. Representing the elements in $S$ by $S_j$ and those in $C$ by $C_i$, a path from $S_j$ to $C_i$ consists in the encryption under some key $K[i,j]$ over some sub-cipher $B$. In other words, the key $K[i,j]$ satisfies the relation
\begin{align}
C_i &= \mathcal{B}_{K[i,j]}(S_j). \tag{0.11}
\end{align}

Let \(d_1\) and \(d_2\) be the cardinality, respectively, of \(S\) and \(C\). If for each \(i \in \{0, 1, \ldots, 2^{d_1} - 1\}\) and \(j \in \{0, 1, \ldots, 2^{d_2} - 1\}\) relation (0.11) is verified, then the 3-tuple composed by the sets \(S\), \(C\) and \(\{K[i,j]\}_{i,j}\) is called a \((d_1, d_2)\)-dimensional (asymmetric) biclique. In the special case where \(d_1 = d_2\), the 3-tuple is simply called \(d\)-dimensional biclique.

Let us suppose we want to construct a \(d\)-dimensional biclique over the last rounds of a cipher \(E\). We begin subdividing the cipher \(E\) in three parts, such that

\begin{align}
P &\xrightarrow{E_1} v \xrightarrow{E_2} S \xrightarrow{\mathcal{B}} C. \tag{0.12}
\end{align}

where \(E_1\) is the subcipher mapping the plaintext \(P\) in an internal state \(v\), \(E_2\) is the subcipher mapping the state \(v\) into another internal state \(S\) and \(\mathcal{B}\) is the subcipher mapping the state \(S\) in the ciphertext \(C\). We want to construct the biclique over the subcipher \(\mathcal{B}\). We choose a base computation, that is the triple \(\{S_0, C_0, K[0,0]\}\), where the key \(K[0,0]\) maps the internal state \(S_0\) to the ciphertext \(C_0\)

\begin{align}
S_0 \xrightarrow{K[0,0]} _\mathcal{B} C_0. \tag{0.13}
\end{align}

We look for \(2^d\) forward differentials \(\Delta_i\), connecting the internal state \(S_0\) to the ciphertexts \(C_i\):

\begin{align}
S_0 \xrightarrow{K[0,0] \oplus \Delta^K} _\mathcal{B} C_0 \oplus \Delta_i = C_i, \tag{0.14}
\end{align}

and similarly, for \(2^d\) backward differentials \(\nabla_j\), connecting the ciphertext \(C_0\) with the internal states \(S_j\):

\begin{align}
S_j = S_0 \oplus \nabla_j \xleftarrow{K[0,0] \oplus \nabla^K} _\mathcal{B} C_0. \tag{0.15}
\end{align}

If the trails of all differentials \(\nabla_j\) do not share any active non-linear operation with the trails of all differentials \(\Delta_i\), then we can find an encryption path which connects any of the \(2^d\) input differences \(\nabla_j\) with any of the \(2^d\) output differences \(\Delta_i\). We then obtain a set of \(2^{2d}\) independent \((\Delta_i, \nabla_j)\)-differential trails:

\begin{align}
S_j = S_0 \oplus \nabla_j \xrightarrow{K[0,0] \oplus \Delta^K \oplus \nabla^K} _\mathcal{B} C_0 \oplus \Delta_i = C_i, \tag{0.16}
\end{align}

for all \(i, j \in \{0, \ldots, 2^d - 1\}\). According to 0.16, \(\mathcal{B}\) connects \(2^d\) internal states \(\{S_j\}\) to \(2^d\) ciphertext \(\{C_i\}\) with \(2^{2d}\) keys \(\{K[i,j]\}\). We now can ask to the oracle to decrypt ciphertexts \(C_i\), for all \(i = 0, \ldots, 2^d - 1\), with the secret key \(K_{sec}\) obtaining \(2^d\) plaintexts \(P_i\), that is

\begin{align}
C_i \xrightarrow{\text{decryption oracle}} _{K_{sec}} P_i. \tag{0.17}
\end{align}

Finally, if one of the keys \(K[i,j]\) is the secret key, then it would map the plaintext \(P_i\) to the intermediate state \(S_j\). In other words, we check if there exist \(i \in \{0, \ldots, 2^d - 1\}\) and \(j \in \{0, \ldots, 2^d - 1\}\) such that
In Figure 0.5 we represent this kind of attack.

We now see which are the results obtained applying this attack to KLEIN (first) and (then) to AES.

KLEIN is a family of lightweight block ciphers proposed by Gong et al. [GNL12] in RFIDsec 2011. It has a fixed block size of 64 bits and supports three different key sizes of 64, 80 and 96 bits. The structure of KLEIN is a typical Substitution-Permutation Network (SPN). The number of rounds depends on key length: we have \( N_r \) equal to 12, 16 and 20 respectively for KLEIN-64, KLEIN-80 and KLEIN-96. It uses a combination of a 4-bits S-box (applied 16 times in parallel) along with AES MixColumn function.

We determined an attack to the 64-bits version of KLEIN using independent 4-dimensional bicliques through a java framework, named Janus, developed by Wenzel et al. [AFL+13] and available at \( \text{https://github.com/janus-framework/janus} \). In order to mount such an attack, we determined a 4-dimensional biclique over the first two rounds of this cipher. We split then the cipher as

\[
P \xrightarrow{B} S \xrightarrow{f} C.
\]

where \( B \) is the subcipher mapping a plaintext \( P \) into the internal state \( S \) (covering the first two rounds) and \( f \) is the subcipher mapping the internal state \( S \) into the ciphertext \( C \).

We then defined a partition of the key space into \( 2^{k-2d} = 2^{56} \) groups of \( 2^{2d} = 2^8 \) keys and setup the base computation \( \{ P_0, S_0, K[0, 0] \} \) as follows:

\[
P_0 = 00, 00, 00, 00, 00, 00, 00, 00,\]

\[
K[0, 0] = 00, 00, 00, 00, 00, 00, 00, 00,\]

and
We then computed the forward and the backward differential trails, respectively \( \Delta \) and \( \nabla \). In the first case we inserted the key difference \( \Delta_i = 00, 00, 00, 0i, 00, 00, 00, 00, \) for \( i = 1, \ldots, 2^4 - 1 \), with respect to \( K[0,0] \) and computed the following \( 2^4 - 1 \) forward differential trails

\[
P_0 \xrightarrow{K[0,0] \oplus \Delta_i} S_i.
\] (0.23)

A similar procedure is applied to compute backward differential trails. We selected the initial key difference \( \nabla_j = 00, 00, 00, 0j, 00, 00, 00, 00, 00, \) for \( j = 1, \ldots, 2^4 - 1 \), with respect to the second round key \( K^2 \), and we determined, using the key schedule algorithm, the related key difference, with respect to \( K[0,0] \) namely

\[
\nabla_j = 0j, 00, 00, 00, 0j, 00, 00, 00.
\] (0.24)

We then computed the \( 2^4 - 1 \) backward differential trails

\[
P_j \xleftarrow{K[0,0] \oplus \nabla_j} S_0
\] (0.25)

for \( j = 1, \ldots, 2^4 - 1 \). In Figure 0.6 we show both forward and backward differential trails.

*Fig. 0.6. Differential trails not sharing bits active in non-linear operation*

The trails computed so far do not share any active non-linear component, and hence, we could define the biclique connecting the \( 2^4 \) input differences \( \nabla^b \) with the \( 2^4 \) output differences \( \Delta_i \), obtaining a set of \( 2^{2^4} \) independent \((\Delta_i, \nabla_j)\)-differential trails.
where

\[ P_j^{K[i,j]} \rightarrow B_S i, \quad (0.26) \]

\[ P_j = P_0 \oplus \nabla_j, \quad (0.27) \]

\[ S_i = S_0 \oplus \Delta_i \quad (0.28) \]

and

\[ K[i, j] = K[0, 0] \oplus \Delta_i \oplus \nabla_j. \quad (0.29) \]

Analysing the part of the cipher not covered by bicliques, looking for the variable \( v \) determining the optimal match, we have that, for KLEIN, we got this match on the rightmost byte at round 6. Figure 0.7 shows the resulting matching trails, where blue cells represent components that should be recomputed. The complexity of the attack returned by this framework is \( 2^{63.17} \) being less than that of the exhaustive search (which is given by \( 2^{64} \)).

After having analysed the attack to KLEIN, we finally determined an attack to the Advanced Encryption Standard.

AES is a Substitution-Permutation Network adopted by NIST in November 2001. It is an iterated block cipher with a fixed block length of 128 bits and supports key size of 128, 192 and 256 bits. The number of rounds \( N_r \) depends on key length: we have \( N_r \) equal to 10, 12 and 14 respectively for AES-128, AES-192 and AES-256.

As we did before, we split the key space \( F_2^{128} \) into \( 2^{128-2-8} \) groups having \( 2^{16} \) keys each. Since we were defining a biclique over the last 3 rounds and since the key schedule uniquely maps a key \( K \) in the round key \( K^8 \), we defined the set of keys with respect to this round. To have a more compact and therefore more easily readable notation, we represented the keys in the matrix form in the following way: if \( K = K_0, \ldots, K_{15} \), then

\[
K = \begin{bmatrix}
K_0 & K_4 & K_8 & K_{12} \\
K_1 & K_5 & K_9 & K_{13} \\
K_2 & K_6 & K_{10} & K_{14} \\
K_3 & K_7 & K_{11} & K_{15}
\end{bmatrix}.
\quad (0.30)
\]

We took the key \( K[0, 0] \) as

\[
K[0, 0] = \begin{bmatrix}
00 & 00 & 00 & 00 \\
00 & 00 & 00 & 00 \\
00 & 00 & 00 & 00 \\
00 & 00 & 00 & 00
\end{bmatrix}.
\quad (0.31)
\]

Injecting the differential

\[
\Delta_i = \begin{bmatrix}
00 & 00 & 00 & 00 \\
00 & 00 & 00 & i \\
00 & 00 & 00 & 00 \\
00 & 00 & 00 & 00
\end{bmatrix}.
\quad (0.32)
\]
**Fig. 0.7.** Merged differential trails.
for $i = 1, \ldots, 2^8 - 1$, with respect to $K[0, 0]$ we got the $2^8 - 1$ forward differential trail represented in Figure 0.8. To compute the $2^8 - 1$ backward differential trail, represented in Figure 0.9, we injected the differential

$$\nabla_j = \begin{bmatrix}
00 & 00 & 00 & 00 \\
j & 00 & 00 & 00 \\
00 & 00 & 00 & 00 \\
00 & 00 & 00 & 00
\end{bmatrix}$$

(0.33)

with respect to $K^{10}$, obtaining the differential

$$\nabla_j = \begin{bmatrix}
00 & 00 & 00 & 00 \\
j & 00 & j & 00 \\
00 & 00 & 00 & 00 \\
00 & 00 & 00 & 00
\end{bmatrix}$$

(0.34)

with respect to $K[0, 0]$.

Since differentials trails for $\Delta_i$ and $\nabla_j$ share no active non-linear bytes, we could combine them in order to determine the 8-dimensional biclique. Therefore the keys $K[i, j]$ are given by

$$K[i, j] = K[0, 0] \oplus \Delta_j \oplus \nabla_j = \begin{bmatrix}
00 & 00 & 00 & 00 \\
j & 00 & j & i \\
00 & 00 & 00 & 00 \\
00 & 00 & 00 & 00
\end{bmatrix}$$

(0.35)

Analysing the part of the cipher not covered by bicliques, we obtained an optimal match at round 5 on the 16th byte of the state before the SubBytes operation. Figure 0.10 represents the forward and the backward match. In this figure, bytes in gray represent bytes that should be recomputed.

The complexity of the attack returned by this framework is given by $2^{126.44}$ which is better than $2^{128}$ (as obtained through exhaustive search) but it is worst than the one obtained by Bogdanov et al. given by $2^{126.1}$.

A description of the methods used can be found in Chapter 2, while in Chapter 3 we provide the entire description of this analysis.
Fig. 0.8. Forward differential trails

Fig. 0.9. Backward differential trails
Fig. 0.10. Merged differential trails
References


