Improved cryptanalysis of an AES implementation

Ludo Tolhuizen
Philips Group Innovation, Research
High Tech Campus 34, 5656 AE Eindhoven, The Netherlands
ludo.tolhuizen@philips.com

1 Introduction

In [1], Chow et al. provide an implementation of AES as a series of lookups in key-dependent tables. The tables depend not only on a key, but also on randomly chosen permutations of the set of 8-bits bytes. The idea is that many pairs of keys and permutations may give rise to the same table, thus obfuscating the key.

The cryptanalysis of Billet et al. [2] shows how to reconstruct the used byte permutations. The most time consuming part of their method deals with finding the used byte permutation up to an affine mapping; it has a time-complexity of at most $2^{24}$, thus essentially cracking the given AES implementation. In this paper, we provide a variation on this part of the attack, reducing the time complexity even further, viz. to at most $2^{14}$.

In the remainder of paper, we first formulate the the task performed in the most time-consuming part of the method of [2] in an independent setting. Next, we outline our algorithm, explain a pre-processing, describe our algorithm in pseudo-code, and provide a complexity analysis.

2 Problem statement and solution

The task performed in the most time-consuming part of the method from [2] can be phrased in an independent setting as follows [2, Thm. 1].

**Theorem 1** Given a set of functions $S = \{Q \circ \oplus_\beta \circ Q^{-1}\}_{\beta \in GF(2^8)}$, given by values, where $Q$ is a permutation of $GF(2^8)$ and $\oplus_\beta$ is the translation by $\beta$ in $GF(2^8)$, one can construct a particular solution $\tilde{Q}$ such that there exists an affine mapping $A$ such that $\tilde{Q} = Q \circ A$.

The premises of the theorem can be rephrased as follows. We are given $2^8$ tables $T_0, \ldots, T_{255}$ (corresponding to the functions in $S$). Each such table has $2^8$ entries, each entry being an 8-bits string. There is an unknown permutation $Q$ of $\{0, 1\}^8$ and an unknown bijection $\beta : \{0, \ldots, 255\} \mapsto \{0, 1\}^8$ such that for each $i, j \in \{0, 1, \ldots, 255\}$ we have that $T_i(j) = Q(\beta(i) \oplus Q^{-1}(\text{bin}(j)))$, where $\oplus$ denotes bitwise XOR-ing and $\text{bin}(j) \in \{0, 1\}^8$ is the binary representation of $j$.

In [2], an explicit algorithm is given for computing a permutation $\tilde{Q}$ as in Theorem 1 with a time complexity at most $2^{24}$. We provide a variation on this algorithm with a time complexity as low as $35 \cdot 2^8 < 2^{14}$. In fact, we present our algorithm in a slightly more general context, viz. for binary strings of arbitrary length $s$, in which case our its time complexity is at most $(4s + 3)2^s$. 
2.1 Outline of the algorithm

As in [2], we will select from $S$ a collection of $s$ functions that span $S$ through function composition. That is, for each $f \in S$, there is a unique $s$-tuple $\Psi(f) = (\epsilon_1, \epsilon_2, \ldots, \epsilon_s) \in \{0, 1\}^s$ such that

\[ f = f_1^{\epsilon_1} \circ f_2^{\epsilon_2} \circ \cdots \circ f_s^{\epsilon_s}, \]

where $f_i^{\epsilon_i} = f_i$ and $f_i^0 = id$.

We will find the functions $f_1, \ldots, f_s$ and, more importantly, the function $\Psi$, in a more efficient way than in [2]. As shown in [2], from $\Psi$ a particular solution $\tilde{Q}$ can be found by evaluating all functions $f \in S$ in the all-zero vector $0$, using the equation

\[ f(0) = \tilde{Q}(\Psi(f)). \tag{1} \]

Note that Lemma 1 below shows that $\{f(0) \mid f \in S\} = \{0, 1\}^s$; so (1) indeed specifies $\tilde{Q}$.

2.2 Pre-processing: labeling functions

Our algorithm becomes much faster if the functions in $S$ are labeled in a particular order. Before stating how we do this labeling, we first prove a simple observation.

**Lemma 1** Let $Q$ be a permutation of $\{0, 1\}^s$ and let $x_0, \alpha, \beta \in \{0, 1\}^s$. If $Q(\alpha \oplus Q^{-1}(x_0)) = Q(\beta \oplus Q^{-1}(x_0))$, then $\alpha = \beta$.

**Proof** If the premisses holds, then, as $Q$ is a permutation, $\alpha \oplus Q^{-1}(x_0) = \beta \oplus Q^{-1}(x_0)$.

In the pre-processing step, we denote the functions in $S$ as $g_0, g_1, \ldots, g_{2^s-1}$, where the ordering of the functions is such that $g_i(0) = bin(i)$.

Lemma 1 guarantees that such an ordering exists. The advantage of performing this labeling is explained by the following lemma.

**Lemma 2** Let $0 \leq i, j \leq 2^s - 1$. Then

\[ g_i \circ g_j = g_k \]

for the integer $k$ with $bin(k) = g_i(bin(j))$.

**Proof** We have that $g_k(0) = (g_i \circ g_j)(0) = g_i(g_j(0)) = g_i(bin(j))$, where the final equality follows from the definition of the labeling. As $g_k(0) = bin(k)$, the lemma follows.

2.3 Description of the algorithm

After the labeling, we apply the following algorithm, where $in\{0..2^s - 1\}$ is a Boolean array, and $\Phi[0..2^s - 1]$ is an array of elements from $\{0, 1\}^s$.

After termination of the algorithm, we can obtain the function $\Psi$ from the equation

\[ \Psi(g_i) = \Phi[i] \text{ for } 0 \leq i \leq 2^s - 1. \]

Combining this with (1), we find that for $0 \leq i \leq 2^s - 1$,

\[ \tilde{Q}(\Phi[i]) = \tilde{Q}(\Psi(g_i)) = g_i(0) = bin(i). \]

That is, for obtaining $\tilde{Q}$ we can use the array $\Phi$ and need not explicitly compute $\Psi$.
Algorithm

\[
\text{inR}[0] := \text{true}; \Phi[0] := 0; \\
\text{for } i := 1 \text{ to } 2^s - 1 \text{ do begin inR}[i] := \text{false}; \Phi[i] := 0 \text{ end}; \\
j := 0; \\
(** \text{The set of functions } g_i \text{ for which inR}[i] = \text{true consists of } 2^j \text{ functions and is closed under function composition. For each } i, k \text{ with inR}[i] = \text{true}, \Phi[i] \text{ is an } s\text{-bits vector, and } \\
\Psi(g_i \circ g_k) = \Phi(g_m) = \Phi[i] \oplus \Phi[k] = \Psi(g_i) \oplus \Psi(g_k) \text{, where } m = g_i(bin(k)). **) \\
\text{while } j \neq s \text{ do begin } i := 1; \text{ while inR}[i] \text{ do } i := i + 1; (** \text{so } i \text{ the smallest index with inR}[i] = \text{false } **) \\
\text{inR}[i] := \text{true}; j := j + 1; \Phi[i] := e_j; (** e_j \text{ is the } j\text{-th unit vector } **) \\
\text{for } k := 1 \text{ to } 2^s - 1 \text{ do } \\
\text{if inR}[k] \text{ then begin } m := g_i[k]; \text{ inR}[m] := \text{true}; \Phi[m] := \Phi[k] \oplus \Phi[i] \text{ end; end.} \\
\]

Explanation For each \(j\), let \(R_j\) be the set of functions for which the \(\Phi\) (or \(\Psi\)) value has been determined after iteration \(j\). The set \(R_j\) consists of all functions of the form \(f_1^i \circ \ldots \circ f_j^j\). In step \(j\), we extend the set \(R_{j-1}\) to the set \(R_j\). Note that

\[
R_j = R_{j-1} \cup \{ f \circ f_j \mid f \in R_{j-1} \},
\]

and \(f_j\) is the function \(g_i\) where \(i\) is the smallest index for which \(\text{inR}[i] = \text{false}\).

2.4 Complexity estimate

The pre-processing requires \(2^s\) steps: we determine \(f(0)\) for each \(f \in S\).

The initialisation of \(\text{inR}\) and \(\Phi\) requires \(2^{s+1}\) steps.

For each of the \(s\) values of \(j\): find \(i\) (\(2^s - 1\) steps) *, set \(\text{inR}[i] = 1\) (1 step), set \(\Phi[i] = 1\) (1 step), and, for each of the \(2^s - 1\) value of \(k\), three operations†, resulting in at most \(3 \cdot (2^s - 1)\) steps.

We conclude that the time complexity is at most \(3 \cdot 2^s + s \cdot 4 \cdot 2^s = (3 + 4s)2^s\).

The complexity reduction as compared to [2] is due to two reasons: first, the pre-processing step makes it easy to find, for every two functions \(f, g \in S\) the label of the function \(h \in S\) such that \(f \circ g = h\). Secondly, we iterate over \(j\) instead of over the set \(S\) as done in [2].

References


---

*In fact, at most \(2^{j-1}\) steps in step \(j\), as \(R_{j-1}\) has \(2^{j-1}\) elements, i.e., \(\text{inR}[i] = \text{true for } 2^{j-1}\) values of \(i\)

†or, to be more precise: one comparison for all value of \(k\), and three operations for at most \(2^j\) elements \(k\)