Advanced Cryptography: Algorithmic Cryptanalysis

DANIEL LOEBENBERGER, KONSTANTIN ZIEGLER

1. Exercise sheet
Hand in solutions until Saturday, 16 April 2011, 23:59h, your code (exercises 1.5) until Saturday, 23 April 2011, 23:59h.

Basic attacks.

Exercise 1.1 (Attack models). (7 points)

In the lecture we saw that there are several different attack models. Each attack model consists of an attack goal (key recovery, message recovery or distinguishing) together with the resources of the adversary (public information only, ciphertext only, known plaintext, chosen plaintext or chosen ciphertext).

(i) Specify the strongest attack model, i.e. figure out which type of adversary is the least powerful one. [2]

(ii) Specify the weakest attack model, i.e. figure out which type of adversary is the most powerful one. [2]

(iii) Put all the attack models in relation to each other, i.e. draw a table with the fifteen possible attack models and put an arrow between two of them if an attack in one model implies an attack in the other model. [3]

Exercise 1.2 (Brute force). (5 points)

To get a better understanding of the amount of work you need to do when employing brute-force cryptanalysis, estimate for which key-sizes you can exhaustively test all keys within a year using your own computer, all computers of a university with, say, 10000 computers, or all computers in the world (there are roughly 2 billion computers out there). You can assume that testing a single key requires exactly one CPU cycle and that each computer runs with 1GHz on average. [5]

Saturation attacks on AES.

Exercise 1.3 (A property of the exclusive or). (3 points)

Consider the set $B = \{0,1\}^k$ of all k-bit binary strings. Prove that for $k > 1$ we have [3]

$$\bigoplus_{b \in B} b = 0.$$

Exercise 1.4 (MixColumns on lambda-sets). (5 points)

5 Let Λ be a lambda-set with exactly one active byte per column. Show that element-wise application of MixColumns on Λ yields a lambda-set with all bytes active.

Exercise 1.5 (The saturation attack on four round AES running). (20+15 points)

Goal of this exercise is to have a full-scale implementation of the saturation attack on four round AES. You can use any programming language of you choice (please refrain from using esoteric language like Whitespace or Brainfuck). We recommend using the open source computer algebra system sage

`http://www.sagemath.org/`

Hand in your *commented* sources as well as a output trace of your program. Note that if your code does not compile, we cannot grade it.

4 (i) You do not want to reinvent the wheel. So find a nice AES library, that gives the possibility to access the round functions of AES and thus allows implementation of attacks on reduced round variants of AES. In `sage` there is the package `sage.crypto.mq` that can be employed, using for example the lines

```
from sage.crypto import mq

myaes = mq.SR(4, 4, 4, 8, star=True,
                          allow_zero_inversions=True,
                          aes_mode=True)
```

1 (ii) Randomly select a plaintext and a key. In `sage` you can use the command

```
plain = myaes.random_state_array()
key = myaes.random_state_array()
```

3 (iii) Implement now a routine `reduced_round_AES` that realized AES with `num_rounds` rounds on the plaintext `plain` with key `key` (note that the last round differs from the inner rounds.

2 (iv) Realize the generation of lambda-sets with exactly one active byte. Each such set consists of 256 AES states, where one byte ranges over all possible elements of $\mathbb{F}_{256}$, and the other bytes are fixed .

(v) Create a function `guess_key` that

1

(a) Creates two lambda-sets Λ_1 and Λ_2.

(b) Applies four-round AES on each element of Λ_1 and Λ_2, respectively, giving two sets D_1 and D_2. [1]

(c) Guess all round-key bytes k_{ij} for $0 \leq i, j < 4$ individually by using the relations [5]

$$c[\ell]_{ij} = \text{SBox}^{-1}(D[\ell]_{i,j-i} + k_{i,j-i})$$

and

$$\sum_{\ell} c[\ell]_{ij} = 0$$

holding for all ℓ. Return `FAIL` if more than one guess fulfills the relations. Note that in `sage`, there is no predefined function for SBox^{-1}, you will have to create it manually by a function like

```
def inv_sbox(myaes):
  res = {}
  for e in myaes.base_ring():
    res[myaes.sub_byte(e)] = e

  return res
```

(vi) Describe how you can reconstruct the key knowing the fourth round key in AES. [3]

(vii) Implement this step. (This requires, at least in `sage`, somewhat reinvention of the wheel, since there are no decryption libraries in the `crypto` package.) [+5]

(viii) Perform statistics on the failure rate of your routine `guess_key`: Determine how often your algorithm returns `FAIL` with just one lambda-set. Do the same with two lambda sets. Did it happen that the algorithm returned a key that was *not* the correct one? [+10]