CS 741: Advanced Network Security and Cryptography Project Report

Improved Key Recovery Attacks on Reduced-Round AES with Practical Data and Memory Complexities[1]

Attack	Data (Chosen plaintexts)	Memory (128-bit blocks)	Time (encryptions)
MitM [11]	8	2^{56}	2^{64}
Imp. Polytopic [26]	15	2^{41}	2^{70}
Partial Sum [27]	2^{8}	small	2^{38}
Square [9]	2^{11}	small	2^{44}
Square [9]	2^{33}	2^{32}	2^{34}
Improved Square [15]	2^{33}	small	2^{33}
Yoyo [25]	$2^{11.3}$ ACC	small	2^{31}
Imp. Diff. [1]	$2^{31.5}$	2^{38}	2^{33}
Mixture Diff. [20]	2^{32}	2^{32}	2^{32}
Our Attack (Sect. 4)	$2^{21.5}$	$2^{21.5}$	$2^{21.5}$

ACC Adaptive Chosen Plaintexts and Ciphertexts

Table 1. Attacks on 5-Round AES (partial key recovery)

[1]

Darshil Desai (160020018)
Abhro Bhuniya (160050017)
Suraj Narra (160050087)
T. Nikhil (1600050096)

Introduction

- AES is probably the most widely studied and used block cipher. Versions with a reduced number of rounds are used as a building block in many cryptographic schemes, e.g. several candidates of the CAESAR competition are based on it.
- We studied and implemented chosen plaintext key recovery attacks on 5-round AES.
- There are 3 relevant parameters: Time (T), Memory (M) and Data (D) based on which attacks have been compared.

Maths behind the Attack

- **Grassi Distinguisher**: Distinguishes a set of 2^32 ciphertexts of 4 round encryption from a set of 2^32 random sequences. This is used as a precursor to the Grassi five round attack.
- **Grassi Attack**: Prepends the above 4 round distinguisher by a round and as input to this round gives carefully crafted mixtures of Plaintexts.

- **Differential Trail over 2-round AES:**

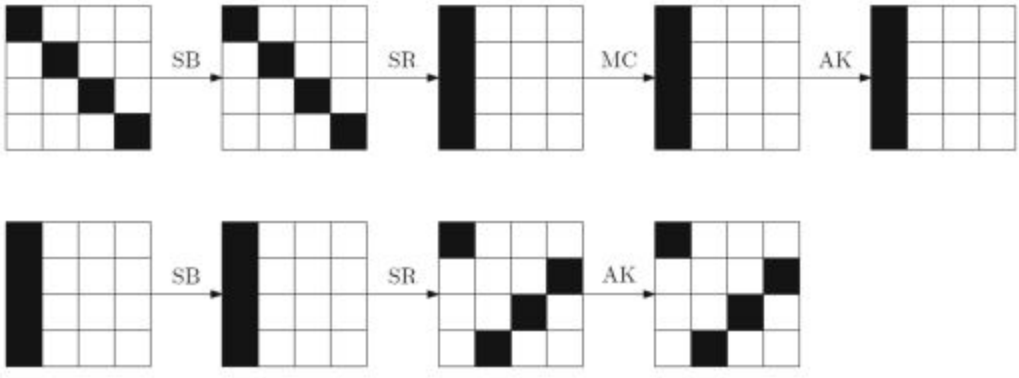

Fig. 1. Differential Trail over 2-round AES.

Consider two plaintexts which are equal in all bytes except for the ones in the i-th diagonal for a certain i = 0, 1, 2, 3, i.e. for the bytes in row j and column i + j for each j = 0, 1, 2, 3 (the index i + j is taken modulo 4). After two AES rounds with fixed random round keys,

$$\begin{bmatrix} A & C & C & C \\ C & A & C & C \\ C & C & A & C \\ C & C & C & A \end{bmatrix} \xrightarrow{R(\cdot)} \begin{bmatrix} A & C & C & C \\ A & C & C & C \\ A & C & C & C \\ A & C & C & C \end{bmatrix} \xrightarrow{R(\cdot)} MC \times \begin{bmatrix} A & C & C & C \\ C & C & C & A \\ C & C & A & C \\ C & A & C & C \end{bmatrix},$$

Assuming the final MixColumns is omitted, the two texts are equal in all bytes except for the ones in the i-th anti-diagonal, that is for the bytes in row j and column i − j for each j (the index i − j is taken modulo 4) by definition of anti-diagonal.

If the final MixColumns is not omitted, certain linear relations - which are given by the definition of the MixColumns matrix - hold between the bytes of the texts that lie in the same column.

- **Theorems and Lemmas used:**
    - If two ciphertexts have equal bytes in d anti-diagonals, then these two texts have equal bytes in d diagonals two rounds before. In other words, a inverse-diagonal set is mapped into a diagonal set two rounds before (assuming the final MixColumns operation is omitted).
    - Given 2^32 plaintexts in the same diagonal set defined as before, consider the corresponding ciphertexts after 5 rounds, that is (p_i , c_i) for i = 0, ..., 2^32 − 1 where c_i = R 5 (p_i) The number n of different pairs of ciphertexts (c_i , c_j) for i = j for which the bytes of the difference $c_i \oplus c_j$ that lie in d anti- diagonals are equal to zero (where 1 ≤ d ≤ 3 and the anti-diagonals are fixed in advance) is a multiple of 8. This gives a bound on good ciphertext pairs and hence a bound on complexity.

- Therefore, if $c_i \oplus c_j$ is equal to zero in d anti-diagonals, then c_i and c_j had the same bytes in their anti diagonals meaning their diagonals 2 rounds before had the same bytes and vice versa. The attack is successful in 2^32 and the improvements discussed below explain how to improve it further.

Grassi's Attack[4]

- In 2017 a new technique (the multiple-of-8 attack) was proposed, and in 2018 Grassi applied a special version of it (the mixture-differentials attack) to 5 round AES.
- Here, we precede the 4 round distinguisher with an extra round before it and we choose plaintexts with all possible values in the diagonals.
- A good ciphertext pair (good_pair) is the one for which in one of the anti diagonals(), all elements match. An anti diagonal is one of the quartets (0, 7, 10, 13), (3, 6, 9, 12), (1, 4, 11, 14), (2, 5, 8, 15) elements when written in row-major order.
- We look for a "good ciphertext pair", and get the corresponding plaintexts. For all 2 32 possible key bytes: partially encrypt (AKR, SB, SR, MC) and create mixtures which are then each decrypted (partially).
- Now encrypt the new mixture plaintexts through 5 rounds and check for equality condition in their anti-diagonals. If the guessed key was the correct key, the condition would satisfy, else a wrong key would result in random permutation of the mixture texts.
- However, its complexity was not better than previous attacks (2^32).

Improvements by the authors

- **Reducing the data Complexity**

  Instead of 2^{32} plain texts used in Grassi's attack, here we use only 2^{24} plain texts chosen arbitrarily from the original 2^{32}. But now it is not guaranteed that the plain texts for mixture quadruple will be there in our data. So, the time complexity increases, since we need to check all pairs of good_pairs to get mixture quadruple.

  Originally 2^{32} plain texts, so, $2^{32}*(2^{32}-1)/2 \cong 2^{63}$ pairs of data, so number of good_pairs are $2^{63}/2^{32} = 2^{31}$. But now there will be 2^{47} pairs of data, so number of good pairs will be 2^{15}. So, now we need to iterate over all 2^{29} pairs of good_pairs to get mixture quadruple. So, in the process of reducing data complexity, it increases time complexity.

  But the probability of getting a mixture quadruple is still pretty high, since we have a total of 7 mixture types, the probability is $1 - (1- 2^{-16})^{7*2\text{^}15} \cong 0.97$

- **Reducing time complexity**

  Now we reduce the time complexity which is increased by previous improvement by discarding some of the key guesses initially.

  Let the diagonal of the key we are guessing is (0, 5, 10, 15) bytes of the key.
  Now let (P_1, P_2), (P_3, P_4) be a pair of good_pairs.
  Then for each guess of key[0], we check if after 1 round encryption of them with the guess,

  $$enc(P_1)[0] \oplus enc(P_2)[0] == enc(P_3)[0] \oplus enc(P_4)[0]$$ holds or not.

  If not, discard that guess for key[0].
  Similarly, do this for key[5], key[10], key[15] with the remaining elements in first column of (P_1, P_2), (P_3, P_4) to discard some of the guesses.

  Now, it is given in paper that the number of guesses remaining will be lesser so that the time complexity is now dominated by the initial loop of discarding the key guesses and is almost equal to that of original Grassi's attack. So, combining this

and the earlier improvement will lead us to have a lesser data complexity with almost same time complexity.

- **Precomputed Table**

We can further reduce time complexity by using a precomputed table for the checking done in discarding key guesses in improvement 2.

We create a 3 Dimensional table, each dimension with 2^8 elements (say table[a,b,c]). We consider quartets of form (0, a, b, c) and in each entry of the table, we store the key byte $k\hat{}$ for which $SB(k\hat{}) \oplus SB(a \oplus k\hat{}) \oplus SB(b \oplus k\hat{}) \oplus SB(c \oplus k\hat{}) == 0$ [7] holds.

This can also be improved by looping over (a, b, $k\hat{}$) instead of (a, b, c, $k\hat{}$), by:
1. Computing $t = SB(k\hat{}) \oplus SB(a \oplus k\hat{}) \oplus SB(b \oplus k\hat{})$
2. The original equality[7] holds only if $c = SB^{-1}(t) \oplus k\hat{}$
3. So, store the $k\hat{}$ in entry table[a, b, $SB^{-1}(t) \oplus k\hat{}$]

If given quartet (x, y, z, w), we consider the quartet (0, $y \oplus x$, $z \oplus x$, $w \oplus x$) while guessing for key[0] and look for table entry table[$y \oplus x$, $z \oplus x$, $w \oplus x$], let it be $k\hat{}$. Then a plausible guess for key[0] will be $k\hat{} \oplus x$

Similarly we do this for other three diagonal elements of key

- **Wise choice of Plain Texts**

We can further reduce all complexities by this improvement, but this also reduces the probability of success of attack.

In this, instead of arbitrary 2^{24} plain texts, we consider the plain texts in which first byte is constant and consider arbitrary 2^{22} plain texts from them.

By applying all the above improvements, the data complexity and time complexity Reduce to almost 2^{22}, which is lesser than many other similar attacks, but now it is a probabilistic attack.

Personal Section

- **Challenges in understanding paper**

  Initially the math seemed daunting but once we watched the conference presentation and the authors explaining their ideas it became easier to understand. The proofs are still out of scope for us but we have grasped the intuition behind the working of the attack.

- **Challenges in design / implementation**

  The improved attacks still requires a high time complexity and the time complexity is with respect to the time per AES encryption. In our implementations of Grassi, the time for the attack was in hours when the key size was 8 bits per element, while grassi recovered the correct key within a minute with key size 4 bits per element. With the improvements, the 8 bit version took about half an hour to run. Also this attack is a probabilistic attack and failed with a high probability for us. All in all, we believe with better computational resources and more time we can successfully complete the attack to recover the key bits.

- **Novel ideas in the paper**

  The idea behind Grassi attack is brilliant. Grassi noticed the evolution of bytes of the states after every round and discovered a vulnerability that would arise when he encrypted 4 similar plaintexts. Based on this he was able to recover key bits.

- **Critique**

  We feel that the source of all brilliance is Grassi's Attack. The paper we referenced first decreased the space of plaintexts at the cost of more time.

After that they mechanically tried to improve the time complexity and this did not feel innovative or brilliant to us.

- **Extensions / enhancements**

  This can be extended to be a 6 round attack, as discussed in the paper[4]. But we couldn't implement it in the given time.

References

1. Our reference paper:
   a. https://eprint.iacr.org/2018/527.pdf
2. Presentation slides of authors:
   a. https://crypto.iacr.org/2018/slides/28838.pdf
3. New 5 round distinguisher by Grassi:
   a. https://link.springer.com/content/pdf/10.1007%2F978-3-319-56614-6_10.pdf
4. Grassi's attack:
   a. https://eprint.iacr.org/2017/832.pdf
5. AES code:
   a. https://gist.github.com/bonsaiviking/5571001
6. Small scale SBox and Inv_SBox:
   a. https://github.com/Krypto-iaik/Distinguisher_5RoundAES/blob/master/AES_smallScale_sbox.h

Code / User Manual

- **File aes.py:**

  Contains code for AES encryption and decryption and is slightly changed from the reference we obtained in order to meet our needs.
  Added an option (self.mc) as an argument to constructor of AES class which can be True or False so as to run Mixed Columns in last round or not.

- **File aes_small.py:**

  Contains code for AES with 4 bits instead of 1 byte for each cell of plain text / key, etc.

The SBox and Inv_SBox are taken from the code of small scale attack written by Grassi[6].
The irreducible polynomial used is x^4+x+1 (0x13).
The round constants for key expansion are obtained by running rcon.py file

- **File rcon.py:**

  Contains algorithm to obtain round constants for key expansion in small scale AES

- **File grassi.py:**

  Contains code for Grassi's original attack. `get_good_pairs()` function returns whenever a good_pair is found.Then loop over all key guesses, checking over all 7 Mixtures. If condition holds, then it is a possible guess of diagonal of key and is printed.

- **File grassi_small.py:**

  Contains code for smaller scale of Grassi's original attack, using 4 bits instead of 1 byte. It uses `aes_small.py` whereas `grassi.py` uses `aes.py`

- **File update.py:**

  Contains code for the improvements proposed in the paper we referenced. `Pre_computed_table` is the table used in improvement 3. `pre_compute()` is the function which populates this table.

  Here `get_good_pairs()` function returns list of all the good_pairs, whereas function with same name in `grassi.py` returns a single pair. `limit` is the variable in which we can change the number of plain texts to use (improvement 4)

  `is_mixture()` is the function used to discard some key guesses as proposed in improvement 2. But here, instead of discarding from all, we are adding from null.

  `update_124()` is the function implementing improvements 1, 2, 4.

  `update_34()` is the function implementing improvements 3, 4. It calls the function `pre_compute()`.

- **File update_small.py:**

  Similar to `update.py`, but uses smaller scale, i.e., 4 bits instead of 1 byte. It uses `aes_small.py` whereas `update.py` uses `aes.py`

---THE END---