Modern Cryptology, Project homework 1 - The Square attack on AES

Vicente Bartual Ferran, Bence Mány

February 2022

1 Description of the attack

The Advances Encryption Standard (AES) is a symmetric block cipher specification established by NIST and it's one of the most important, if not the most, ciphers available today. In this assignment, we'll use AES-128, which uses a 128 bits long key to encrypt messages. In the standard specification, the algorithm consists of 10 rounds. In each round the following steps are applied: SubBytes, ShiftRows, MixColumns and AddKey. The last round is an exception because at that point the MixColumn operation is dismissed.

The goal of this project is to implement the Square Attack on the AES-128, reduced to 4 rounds. To describe the attack we need first to define what a Λ-set is. Basically, it is a set of 256 messages with the same values in every byte except for the first, which has a unique value in each message (from 0 to 255). Due to the structure of AES, after three rounds of encryption, these messages meet the following rule: taking any byte position, if we XOR the 256 bytes of each message at that position, the result is 0. The last-round key can be recovered by guessing each byte of the key, reversing the fourth round and checking against this rule. Finally, the original key can be recovered by inverting the key schedule algorithm.

Therefore, the attack is built up by the following steps:

1. Encrypt a randomly chosen Λ-set using the 4 round AES.
2. Reverse the last round by guessing a byte of the last round key. This is done by reversing each possible byte for the given position and then checking if the guess was correct.
3. Remove false positives. Possibly, multiple bytes will comply with the rule at a given position,. However, only one solution should be possible. These fake solutions can be eliminated by performing the attack with a new Λ-set with the same key. The correct guess should still be a solution but the others would change.
4. Continue the same process for each position until the whole last-round key is assembled.

5. Recover the original key by reversing the key schedule.

2 Implementation

Our implementation is distributed across the following files or modules:

1. **aes**: Our implementation of the AES-128 encryption algorithm. Given a plaintext, a key and the desired number of rounds, it produces the corresponded ciphertext.

2. **lambda_set**: Generates a new random Λ-set.

3. **square_attack**: This module implements two functionalities: Firstly, it checks a guess of one byte of the last-round key against a given encrypted Λ-set. Secondly, once the last-round key is recovered it provides a function to retrieve the original cipher key by performing the inverse of the key schedule.

4. **main**: The core of the program which uses the rest of the modules to perform the attack, as many times as necessary to rule out false positives from the guesses of the key.

The program starts with generating a random key and a random Λ-set which is then being encrypted with the AES-128, using the generated key.

In the next step, a for loop goes through all bytes of the encrypted Λ-set, trying to guess the correct byte for each position. The correctness is being checked by the **check_guess** function. Whenever a match is found, the given byte is then stored in an array (**guesses**).

As it was mentioned earlier, multiple guesses can give the same result therefore it's not obvious which one is the real solution. Thus the **guesses** array stores each positive guess for each byte of the key. To get the real solution, the false positive guesses have to be eliminated. It is done by generating a new Λ-set, encrypting it and performing the attack again. This process will result in a new **guesses** array, which has to contain the real solutions again for each byte and possibly some new fake ones. Eventually the two arrays are being compared and the element which can be found in both arrays will be kept, the others will be eliminated.

There is an extremely low chance that the same false positive guess appears in the following rounds but that case has to be handled as well. Therefore a while loop was implemented to repeat the process of generating and encrypting new Λ-sets, creating guesses, comparing them with the previous guesses and eliminating the fake ones until the solution contains only one guess for each position. In that case the round-key can be assembled. After reverting the key schedule to recover the original key, the attack is complete.