UNIVERSITÉ LIBRE DE BRUXELLES, UNIVERSITÉ D'EUROPE

ULB

FACULTÉ DES SCIENCES
DÉPARTEMENT D'INFORMATIQUE

Square Attack on reduced-round AES Block Cipher

Computer Security –

INFO-H-405

Project 1

Mark Skuratov
Viktoriia Aleksenko
Ekaterina Asfandyarova
Victor Mwerekande

1. Justification of the choice of language (C or Java)

According to the task of the project the program has to be written in C++, because C++ already has a library that implements AES. But there is another implementation of this algorithm in Java, thus it was possible to implement project on Java instead of C++. We've chosen for the following reasons:

- syntax of Java is generally easier. As a result, code becomes simpler and its readability and comprehensibility are improved;
- process of memory management in Java is easier than in C++, C++ needs to constantly clean the memory after working with objects, while Java does it automatically;
- Java is a modern language, created to be more efficient and at the same time the aim of its creators was to simplify syntax of the new language.

2. Description of the problem

Given data (specifications): user enters the secret key (in hex) and indicates one of the 16 (bytes) positions for beginning the recovery process.

End product (result): displays the key (AK4) computed by the program.

Algorithm: In the 4th-round of AES - we assume a value for a byte from AK4 (specified by the user) and begin partially decrypt: AK4, SR, SB - reaching 3rd round. This is done 256-times, for this byte of all 256 plaintexts. These values are XORed with each other. The result is checked and if the received byte is zero, assumed byte can be a byte of the key AK4. If we get a zero for several assumption (more than one candidate for key byte), then it will be necessary generate new set of 256 plaintexts and repeat this operations again. Then the same is carried out for all other positions of AK4. The algorithm is publicitely available, the variables for it can be found, for example, here http://en.wikipedia.org/wiki/Advanced_Encryption_Standard

3. Implementation

GUI:

1. Button "Generate Key" - output value of secret in the input field (manual input is also available). It works method SquareAttack.generateKey.
2. Button "Set chosen key" - checks the length of the key and the correct format (it must be in hex) and displays the selected key in the output field on the right part of window. It works method SquareAttack.setKey.
3. Button "Show Original KA4 key" - displays original key AK4 in hex for user reference in the same output field. It is calculated from inputed secret key (for example, KA4: 09 86 30 8f ff 87 09 c3 d1 b5 e2 9c 71 2c ab 56). It works method AES.getRoundKey for fourth round.
4. 16 buttons (in square 4 by 4) represent 16 bytes of AK4 that will be found by algorithm - user opens position after position in any order. Here program generates 256 plaintexts using method SquareAttack.generate256Texts, encrypts it (SquareAttack.encryptTexts) and implements attack for this position with method SquareAttack.attackKeyByte. User can see recovered byte in this position and description of the processus in output field.

Software:

We used the following specific libraries: aes.AES (class found in the Internet), aes.SquareAttack (class that we have developed).

Main program files:

MainFrame.java - here we define the fields and buttons in the GUI (class MainFrame). For key generation class SquareAttack method generateKey is used, as well as its methods: SquareAttack.setKey, SquareAttack.generate256Texts, encryptTexts, attackKeyByte (for details please see class definition);

AES.java - describes class AES, variables of cryptographic algorithm (Nb, Nk, Nr, sbox, invSbox, rcon) and support functions for encryption (deletePadding, getRoundKey, xorFunc, generateSubkeys, subWord, rotateWord, addRoundKey, subBytes, invSubBytes, invSubByte, shiftRows, invShiftRows, invMixColumns, fFMul). Additionally here there are methods which work directly with blocks (encryptBloc and decryptBloc) and with full encryption (encrypt and decrypt).
SquareAttack.java - implements 4-rounds AES attack via the following methods:
- setKey - displays the chosen key;
- generate256Texts - generation of 256 plaintexts with one different byte;
- encryptTexts - encrypts plaintexts with selected key, using the class AES.encrypt;
- attackKeyByte - recovering (guessing) key AK4 using the class AES.invSubByte;

5. Computational complexity:
- Number of encryptions: For everyone from 16 items we encrypt 256 plaintexts, so a total number will be 4096 cycles performed (2 power 12) (If we don't have several candidates for the correct key. In this case, it must add 256 encryptions for every additional verification).
- Partial decryptions for every assumed byte – 256 * 256 times, because we have 256 plaintexts and we have to check 256 bytes as the key byte candidate. We have 16 bytes, so totally it'll be 2^{20} operations. But as in the previous paragraph, it depends of the presence of controversial positions (256 partial decryption for every additional verification of candidate).
- Data requested - 256 plain texts are used to recover each byte of AK4 key (15 bytes are generating 1 time for each set of 256 plain texts and 1 byte called active byte differs from the same byte from the other texts of the set). The 256 plain texts are stored in form of the two-dimensional bytes array 256 by 16. The AK0 can be generated or inputted by user in hex format and it's stored in form of 16 byte array. The original AK4 key can be showed only after first encrypting, because all the keys are generating only during the encryption process.
- Memory usage - the implemented algorithm during its work stores all the keys (5 * 16 bytes = 80 bytes), 256 generated texts (256 * 16 bytes = 4 Kbytes), 256 cipher texts (256 * 16 bytes = 4 Kbytes) and 256 bytes to store the results of partial decryptions. Totally we need up to 9 Kbytes of memory taking into consideration the fact that we need to store local variables (for example, counters), intermediate values and bytes-candidates.