hastim@oregonstate.edu
Mohan Krishna Hasti

Homework 4

1. **Please complete this assignment (200 pts total) and submit your report/program code on Canvas (all files compressed in one .zip without the .bin/.txt files) 1. A program needs to be developed such that fault attacks on a RSA-CRT implementation can be exploited to reveal the original RSA primes of a 1024 bit RSA key. The fault was previously generated during one CRT exponentiation and caused a faulty $s'$of the RSA-CRT while computing the digital signature. The wrong result $s'$is the important input parameter for the program. Other input parameters for the program are the public RSA parameters, moduluns N and exponent e, in addition to the correct result s and the message m to be signed. Two different use cases should be considered. (100 pts)**
   a) **the RSA modulus N and both the correct result s and a faulty result $s'$of an RSA-CRT are used (Bellcore) (50 pts)**

      q =
      3258435542900206882011406659252794438455437084574019274851099246277846549533384206516527557992933685589136885601511045940696233312726738700693963698488 3

      p =
      3343014699932574834486400062724942446282921089048903672549693237411543833898659799117543113600623364982724751633946245990580023392774069384346467258938278139

   b) **the RSA modulus N, the public exponent e, the faulty result s', and the message m are used (Lenstra) (50 pts)**

      q =
      4803764306868804668773207707659330706566295853417667728492123829599941596734986477315094986005520233015928819899019319589119186080322011736805260012300 9

      p =
      2847964824927637344993692960955283032782855967321159107889725459612329395728793747044913299774411642193907163140863799636093590460390137004726094068743430279

2. **All needed input parameters are provided in two separate ASCII files. The hexadecimal representation starts with the most significant bit and ends with the least significant bit. Determine the RSA primes from the input file fault1.txt and fault2.txt. Your program should output the original RSA primes. Note: you do not have to parse the files but can copy-paste the respective values into your script. 2. Conduct Differential Fault Analysis (DFA) on AES-128. You are given a set of three files ptext.bin, ctext.bin, and ftext.bin, corresponding to the plaintext, ciphertext, and (potentially) faulty ciphertext respectively. Note that faults are always injected before the "MixColumns" operation in the 9th round of AES, and each fault only impacts at most one byte. (100 pts)**
   a. **Having a reliable fault injection setup is critical to successfully executing DFA. From the glitch data provided, how many glitches are successful? Does that seem like a high success rate? (5 pts)**

hastim@oregonstate.edu
Mohan Krishna Hasti

There were 4 glitches in the given data at 0, 13, 10, 7, which corresponds to the first column. Yes, it's a high success rate of about approx. 98%, as the glitch was introduced in the first element and it propagated to an entire column.

b. To perform the DFA attack described by Piret and Quisquater, we will need to find multiple ciphertext/faultytext pairs. How many total pairs will we need? By parsing the provided files, find enough pairs to complete the attack and output them here in hex format. Can anything happen that might cause you to need more pairs? (15 pts)

We need atleast 8 pairs to complete the attack In case of a full attack as the glitch in in 4 columns, but since there is 2% chance of getting a failure as in this case, the 2nd column key could not be found with the first pair so had to test with several pairs to find the correct key. There might be chances to get a different key. Due to some external factors or due to wrong fault injection, fault can be injected in a wrong place and might not be useful.

c. Simple Piret and Quisquater DFA: We will provide pairs with glitch in the first byte. Recover 4 bytes of the key (30 pts)

Key = 168, 138, 164, 45

d. Full Piret and Quisquater DFA: using the provided encryption/glitch data, recover the entire round 10 key (40 pts)

Key = 168, 138, 164, 45, 53, 73, 46, 171 ,93, 213, 55, 198, 170, 35, 50, 172

e. Recover the original (first round) key and use it to decrypt the following (hex encoded) secret message: 2a92fc6ad8006b658f49062c2843ad99 (10 pts)

b'\xa7\xe1\xb1\nT\xce\xb6\xdb\xeb!\xcdtl\xa7yk'

Please write all your programs in one of the following languages/environments: Python/Jupyter, Rust, C/C++, Matlab/Octave, Java. Your .zip file should contain your code, instructions how to make it run (if needed), the figures, a report, etc.