On the security of AES

Georg Krause[1], Anthony Tran[1]

[1]Encryption Technology Pte Ltd, Singapore
info@encryption-technology.com

***Abstract*—We observed a behaviour of the AES standard that allows to construct very efficient side channel attacks with little effort. We found that an encryption does not use the elements of the S-Box evenly and we called the unused locations “Holes”. Based on the knowledge of holes we could develop algorithms that reconstruct the keys using known plaintext or ciphertext only attacks. Finding the holes is a simple process, which can easily be automated. Strategies will be recommended to counter the attack, which restores the original strength of AES.**

***Index Terms*—AES, S-Box**

I. Introduction

We define Holes as elements of an S-Box which are not used during an encryption. We found them while studying the effect of swapping contents of S-Box locations when we worked on evolution based algorithm design. Swapping the content did not change the encryption result in our situation. This means that we found 2 S-Box locations that are not used during the encryption rounds and during the round key generation.

II. Attacking mechanism

A. Finding all holes

First of all, it is required to find all Holes using a systematic approach. Changing the input data for the encryption, but using the same key (let’s assume the key is unknown) creates another set of holes. Our approach during the research stores the encryption result of the unmodified S-Box first. We then need to modify the content of all S-Box location, one at a time. If the encrypted result changes, it means the S-Box location content was used. In cases where no change is detected, we found a hole. These locations and the related plaintext must now be stored for later use.

Modifying the content was simply done by inverting all bits of the content. In our program, this is a XOR with the value 255. After having all locations for this key-data pair found the hole locations can be marked in a 255 bit long string. The result is a 32 byte string, which is used as input to the key reconstruction. Each input value lead to a relatively huge number of holes, on average around 80. Between 20 and 40 such hole sets are needed to succeed the attack and reconstruct the key to 100%. This is as little as 640 to 1280 byte.

B. Reducing key space

This is the basis of the attack. To understand the attack, knowledge of the detailed AES standard is required as provided in FIPS-197. Let’s first show how the basic principle works.

We start with a known plaintext attack, after that, we will show a ciphertext only attack.

Our example software uses AES-256, this means the key is 32 byte long, twice as long as the input data block. The round key generation in the AES standard shows that the first 16 byte of the key are used as the first round key and the second 16 byte will be the next round key. Initially, the first 16 input byte are bitwise XORed with the first-round key on a byte-by-byte basis. The result are the locations within the S-Box, which is used to replace/substitute the input byte for the rest of the encryption process.

$$SBox\_location = input\_byte \oplus key\_byte \qquad (1)$$

Where $\oplus$ is XOR logical operator. Using XOR characteristic, we have:

$$key\_byte = input\_byte \oplus SBox\_location \qquad (2)$$

We do not know which S-Box locations are used and therefore cannot calculate the key byte. However, we know which S-Box locations cannot be used (they are the holes). This leads to key bytes that could not exist at this byte location (refer to Equation (2)).

We use for each key byte an array of 256 byte size, each element containing a possible key byte from 0 to 255 (0x00 to 0xFF). Since x = input_byte $\oplus$ Hole_location, x defines an impossible key byte. All calculated of x for a hole set are then eliminated from the array of possible key bytes. This calculation is repeated for every of the 16-input byte of the state.

C. Calculating the first round-key

Our result shows that even with one set of holes, the key space for the first round-key is dramatically reduced. Without the knowledge of holes, the key space is 256^{16} for 16 key byte each possible of holding 256 values. Now, assuming we have found 80 holes, we have reduced the key space to 176^{16}. This is is significantly lower. The relation is 3.4×10^{38} to 8.4×10^{35} or we can say the key space is reduced from 128 to 118 bit. Moreover, each additional set of holes reduces the key space further.

In our reference program, we got the following results for the first round-key using the first set of holes:

- Number of holes: 84

- Holes: 02-08-0B-0D-0E-11-12-13-16-19-1A-22-26-27-29-2B-2C-2D-2E-33-35-36-37-38-3C-42-44-46-4B-4D-55-56-5B-65-6B-6E-6F-74-77-79-7A-7C-84-89-8C-8F-91-93-94-96-98-9A-9B-9E-A1-A6-AC-AE-B1-B2-B4-B6-BB-BE-C8-CC-D1-D3-D4-D7-D8-D9-DA-DB-DD-E2-E3-EB-EE-F1-F7-FD-FE-FF
- Number of possible key byte for each key location: 172
- Key space: 5.8×10^{35}

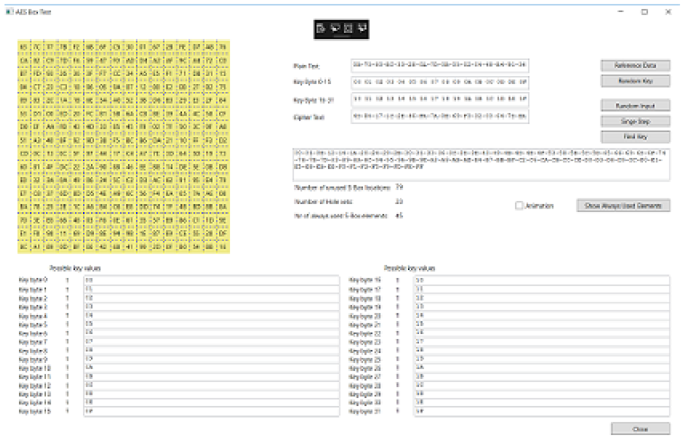

Fig. 1. Screenshot from our key finding program

Figure 1 show how the key is found after additional sets of holes were applied, using different input values for the same key.

D. Calculating the second round-key

The second round-key is reconstructed after the first round-key is known. The same plain data and the same hole sets already used for the first round-key can be applied to the second round-key search. The input data for the xor equation are now the output data of the first encryption round. These can be calculated, because the first round-key is already known. In few cases the second-round key is not 100% found. This means that for a key byte more than one possible values are left over. In such cases a few more hole sets must be evaluated. After the second-round key is found all ciphertext used with this key can be decrypted, because first and second round-key are the two parts of the 256-bit encryption key.

III. Usability in real-world situation

A. Requirements to implement the attack

This approach to break AES implies a set of requirements.

Firstly, hole sets must be collected, together with the known plaintext. This can be achieved in different ways, many of them are more efficient, than the technique used by the authors. Our implementation is simple, but needs 256 encryptions for each hole set. This can be easily optimized to one encryption. Implementations in hardware are possible with little cells required and no extra processing delay. Collecting the holes in a software implementation requires a prepared code or some malicious code that can modify a system during runtime, maybe just for the time of the attack.

Secondly, a side channel is required to deliver the hole sets to the one who runs the analysis. It is up to the reader to imagine methods for delivering this small set of information. We believe this is only limited by the creativity of the attacker.

Lastly, a plaintext must be known, related to the hole set. It is not necessary that a complete 16 byte plaintext block is known. The attack works also when only fragments are know. Then, just more sets are required. Examples 0f known plaintext can be networking packet headers. Header structures of data used by application programs, copyright message text, and so on.

B. Approaches

1) Guessing the plaintext: Can we do better? Yes, we can! One of the more difficult tasks is to know or guess plaintext, when only the ciphertext is seen. Now, the attack works also the reverse way, just a little bit more effort in the reconstruction of the round key. If we can use the ciphertext instead of known plaintext, the attack is reduced to provide the hole set by the side channel.

2) Cyphertext-only attack: A ciphertext-only attack is a little more effort in the analysing program. During decryption, the round-keys are applied in the reverse order. The second to last and the last round-key are the one of interest for us. We must work our way back through all encryption rounds. Hole sets need to be converted to match the inverse S-Box used for decryption. With knowledge of a hole set and the corresponding byte from the ciphertext, the unused key bytes can be calculated. After applying enough hole sets the second to last round-key will be found. The last round-key cannot be found using this method. This is solved by reverse operations applied to the round-key generation. If we know round-keys 2 and 3, it is an easy task to reconstruct round-key 1. After that the complete key is known. A normal PC can perform all the required operations in seconds. What, when a set of holes is known, but the relation to the ciphertext is unknown. This requires multiple analysis attempts, where box elements that are never in a hole identify the correct relation between ciphertext blocks and hole sets.

3) Round-key generation attack: The round keys are expanded from the original key. During this expansion the S-Box of the system is used to provide a non-linear function. When the round keys are calculated on demand, meaning every time when a block is encrypted, then the box elements used for the round key cannot be a hole. While holes are different, when input data are changed, the box elements used for the round key generation are static per key. This can be easily seen when multiple box sets are calculated. The average number of static S-Box elements is around 40. This knowledge may help to attack the round key generation. Further investigation is here required.

IV. Protection

A. Software level

There are several strategies available to protect against attacks based on S-Box holes. For Software (SW) implementations running on a connected PC, less protection is available, because S-Boxes can be easily found within main storage. One strategy is to apply many more encryption rounds, that holes do not exist anymore. Investigation how many encryption rounds are required are planned for the near future.

B. Hardware level

Hardware (HW)-implemented S-Boxes, which cannot be modified by SW, are a better protection. If it is possible to protect the boxes such that third parties cannot know their content, proprietary boxes are another solution. In most cases, this requires a HW implementation. For architectures that apply very frequent key change (every 1-5 encryptions), it is hard to collect enough hole sets.

In systems where HW or SW that have such a technology inbuilt by designer (chips, CPUs) it is difficult to detect such an attack. The amount of data to describe a hole set is small. An average of 5-6 kilobyte in uncompressed format is sufficient. For example, a 100 MB network transfers easily 6,000 packets in a second.

V. Conclusion

As this attack shows, it is not sufficient to use just encryption. Additional ongoing considerations and research is required to apply encryption in a secure way. An obvious conclusion is that SW is much more exposed to attacks and real protection requires HW implementations of encryption algorithms.

References

[1] "Announcing the ADVANCED ENCRYPTION STANDARD (AES)", Federal Information Processing Standards Publication 197 (FIPS-197), November 26, 2001.