Contents

1 Related-key attacks on modern block ciphers

1.1 Description of the related-key attack on AES

In the related-key scenario it is allowed to inject difference into the key, and not only into the plaintext as in pure differential cryptanalysis. To use the attack the information about the encryption key is unknown, but an attacker can control the difference of key pairs.

Local collisions in AES-256 are best understood on a one-round example (Figure 1).

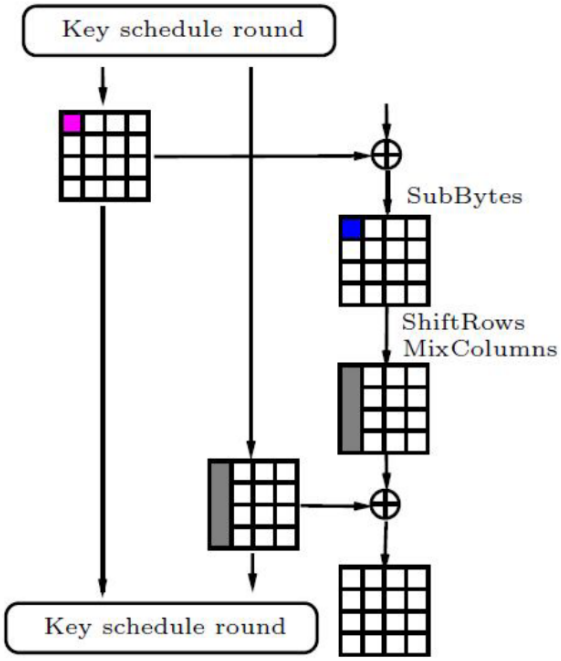

Fig 1 Local collisions in AES

Here one active S-box is needed and five non-zero byte differences in the two subkeys. These five bytes split into two parts: one-byte disturbance and four-byte correction.

Due to the key schedule the differences spread to other rounds. Most of the AES key schedule operations are linear, so a sequence of several consecutive round key values can be viewed as a codeword of a linear code. This is a particularly case when a trail does not have active S-boxes in the key schedule.

Let figure out how to build an optimal trail for the key recovery attack. Typically, a trail is better if it has fewer active S-boxes. Disturbance differences must form a codeword that has a low weight. Simultaneously, correction differences must also form a codeword that sums into the key schedule codeword. In further trails, the correction codeword is constructed from the former one by just shifting four columns to the right and applying S-box and the MixColumns operations. Synchronization is simple since the injection is made to the first row, which is not rotated by ShiftRows. Otherwise, the task of synchronizing two codewords would have been much harder and would have led to high-weight codewords.

An example of a good key schedule pattern for AES-256 is depicted in Figure 2 as a 4.5-round codeword.

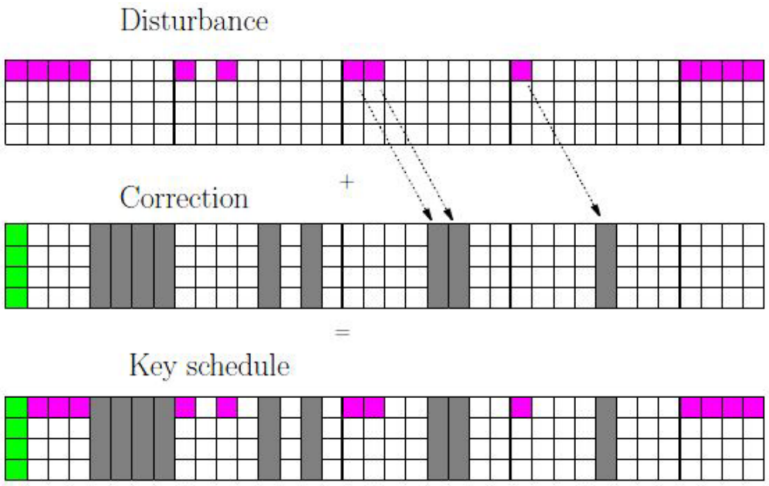

Fig 2 AES-256 key schedule codeword (4.5 key-schedule rounds)

2 A prospective SPN-based block cipher

2.1 General parameters

The encryption algorithm can be used with different types of input data. It supports length blocks of 128, 256 or 512 bits. The length of the key can be also 128, 256 or 512 bits. The internal state of the cipher can be represented as a matrix of bytes. The matrix dimension is $8*N_b$ bytes. In Table 1 acceptable combinations of block and key sizes are represented.

Size of block, bits	Supported key size, bits
128 ($N_b = 2$)	128, 256
256 ($N_b = 4$)	256, 512
512 ($N_b = 8$)	512

Table 1 Acceptable combinations of block and key sizes

Number of rounds depends on the size of block and key that are presented in Table 2.

In this paper the 128-bit version of the cipher is considered. Presented information in the next chapters is concerned only about this version if another is not mentioned.

Size of block, bits	Size of key 128 bits	Size of key 256 bits	Size of key 512 bits
128 ($N_b = 2$)	10	14	–
256 ($N_b = 4$)	–	14	18
512 ($N_b = 8$)	–	–	18

Table 2 The number rounds for different versions of the cipher

2.2 Basic transformations

In the presented algorithm four basic transformations are used:

- key addition;

- bytes substitution;

- shift rows;

- mix columns.

These transformations are the basis of entire high-level structure.

2.3 Key expansion scheme

Let K_M be the main key of encryption and (K_1, K_2, ..., K_m) are the round keys that are generated by the key expansion scheme.

The cipher is based on an SP network similar to the AES cipher

$$Cipher_{K_M} = \prod_{i=1}^{N_r} \theta \circ \gamma \circ \sigma_{k_i},$$

where σ_{k_i} – round key adding;
γ – nonlinear layer (bytes substitution);
θ – linear layer (mix columns, shift rows);
N_r – amount of rounds in block cipher.

The key expansion scheme contains the following two steps:

1. Computing of an intermediate value K_t based on a master key K_M and constants;

2. Computing of round keys (K_1, K_2, ..., K_m) based on a master key K_M, an intermediate value K_t and constants.

The computation of intermediate value K_t is depicted in Figure 3. The algorithm contains all those transformations that are used in the encryption cipher. This reduces implementation complexities because all basic operations are the same.

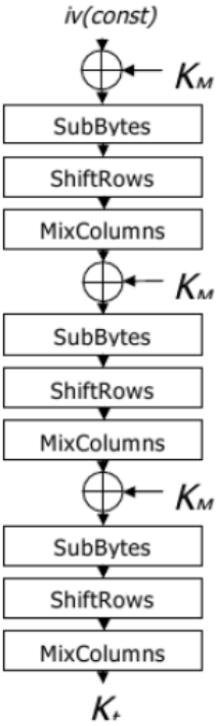

Fig 3 Computing of K_t

General algorithm for the round keys generation is represented in Figure 4.

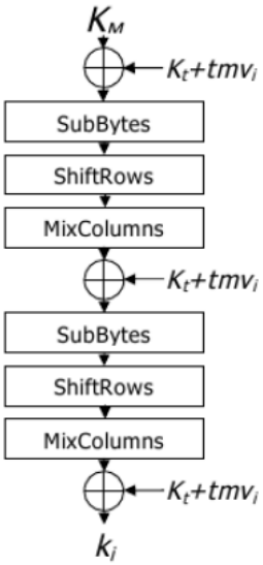

Fig 4 The computation of round keys

In the above scheme (Figure 4) the transformations similar to the scheme of computation K_t and the encryption scheme are used. This reduces the complexity of implementation. According to the scheme has several properties, such as:

1. One-way mapping: having an encryption key it is very easy to generate round keys, but

having one or more round keys it is computationally very difficult to retrieve the encryption key or another round key;

2. Non-linear dependence between each bit of encryption key;
3. Statistical properties of the key schedule were verified by the NIST statistical test suite;
4. Simple implementation (based on the cipher round transformations only), good key agility and possibility to generate round keys in direct and reverse order with the same computational complexity.

3 Absence of Key Commitment in AE schemes

3.1 Polynomial MAC based schemes

To generate valid tags with such an authentication scheme, we have to ensure that the given ciphertext leads to the same tag being computed under K_1 and K_2. We fix all ciphertext blocks apart from a single block $C[j]$, which gives us the following equation:

$$s_1 + C[j] \cdot r_1^{m-j} + \sum_{i=1, i\neq j}^{m} C[i] \cdot r_1^{m-i} = s_2 + C[j] \cdot r_2^{m-j} + \sum_{i=1, i\neq j}^{m} C[i] \cdot r_2^{m-i}. \tag{1}$$

All the variables here are known to the adversary, therefore the equation (1) can be rearranged to isolate $C[j]$

$$C[j] = \frac{s_1 + s_2 + \sum\limits_{i=1, i\neq j}^{m} C[i] \cdot r_1^{m-i} + C[i] \cdot r_2^{m-i}}{r_1^{m-j} + r_2^{m-j}}, \tag{2}$$

which fully determines C and T.

Instead of computing the polynomial MAC over the ciphertext, AES-GCM-SIV computes it over the plaintext, which is then XORed with the nonce and encrypted to get the tag T. T is then further used as the first counter block for encryption. In this case we will first pick T, which fixes the corresponding key streams S_1, S_2. Next, we decrypt the tag with K_1, K_2 and XOR the nonce to obtain T_1 and T_2:

$$T_1 = \sum_{i=1}^{m} M_1[i] \cdot r_1^{m-i} \qquad \text{and} \qquad T_2 = \sum_{i=1}^{m} M_2[i] \cdot r_2^{m-i}. \tag{3}$$

Additionally, we have the condition that the ciphertext should be equal after adding the key streams, therefore we get m equations of the form

$$\begin{cases} M_1[1] + S_1[1] = M_2[1] + S_2[1] \\ M_1[2] + S_1[2] = M_2[2] + S_2[2] \\ \qquad\qquad \dots \\ M_1[m] + S_1[m] = M_2[m] + S_2[m] \end{cases}. \tag{4}$$

In total this gives us $m+2$ linear equations in $2m$ variables (the plaintext blocks), which we can find a solution for if $m > 1$. In general, this still gives us a lot of freedom in the message blocks as for longer messages we can fix parts and still find a solution to the system of linear equations.