Basics of AES

The block cipher AES (short for **advanced encryption standard**) replaced DES. By now, it is the most important symmetric block cipher.

1997: NIST requests proposals for AES (receives 15 submissions) [very different from how DES was selected!]

2000: Rijndael (by Joan Daemen and Vincent Rijmen) selected (from 5 finalists)

https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf

- 128 bit block size (as per NIST request)
- The key size of AES can be 128, 192 or 256 bit. The corresponding choices are referred to as AES-128, AES-192 and AES-256, and have 10, 12 and 14 rounds, respectively.
- AES-192/256 is first (and only) public cipher allowed by NSA for top secret information.
- No known attacks on AES which are substantially better than brute-force.

Attacks better than brute-force known if the number of rounds was 6 (instead of 10) for AES-128.

- Unlike DES, AES is not a Feistel network.

While for a Feistel network, each round only encrypts half of the bits, all bits are being encrypted during each round. That's one indication why AES requires fewer rounds than DES.

Internals of AES

Each round consists of 4 **layers**. Each layer takes 128 bits input and outputs 128 bits in a reversible way (so that we can decrypt as long as we know the key).The 128 bit state consists of 16 bytes. These 16 bytes $c_{0,0}, c_{1,0}, c_{2,0}, c_{3,0}, c_{0,1}, \ldots, c_{3,3}$ are often arranged in a 4x4 matrix as

$$\begin{bmatrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} \end{bmatrix}.$$

Each byte is identified with an element of $\mathrm{GF}(2^8)$.

Example 141. $(0000\ 0101)_2$ represents the element $x^2 + 1$ in $\mathrm{GF}(2^8)$.

The 4 layers are:

- ByteSub

  each byte gets substituted with another byte (like a single S-box in DES); provides confusion and guarantees non-linearity of AES

- ShiftRow

  the 16 bytes are permuted (like a P-box in DES but on bytes, not bits); provides diffusion

- MixCol

  the 4x4 matrix is linearly transformed; provides diffusion

- AddRoundKey

  the state is xored with a 128 bit round key

Armin Straub
straub@southalabama.edu

Slight deviations. Before the first round, AddRoundKey is applied with the 0th round key (which equals the AES key). Otherwise, our first step would be ByteSub, which wouldn't have any cryptographic effect since the plaintext bytes would just be changed in a fixed manner (no key involved yet).

Also, the last round has no MixCol layer. This has the effect that decryption can be made to look very much like encryption (see Section 5.3 in our book for the details).

ShiftRow

The layer **ShiftRow** permutes the 16 bytes $c_{0,0}, c_{1,0}, c_{2,0}, c_{3,0}, c_{0,1}, \ldots, c_{3,3}$ as follows:

$$\begin{bmatrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} \end{bmatrix} \mapsto \begin{bmatrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\ c_{1,1} & c_{1,2} & c_{1,3} & c_{1,0} \\ c_{2,2} & c_{2,3} & c_{2,0} & c_{2,1} \\ c_{3,3} & c_{3,0} & c_{3,1} & c_{3,2} \end{bmatrix}$$

MixCol

Again, arrange the 16 bytes as a 4×4 matrix with entries in $\mathrm{GF}(2^8)$. The **MixCol** layer transform this 4×4 matrix by multiplying it with another, fixed, 4×4 matrix:

$$\begin{bmatrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} \end{bmatrix} \mapsto \begin{bmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{bmatrix} \begin{bmatrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} \end{bmatrix}$$

Example 142. For instance, the new byte at the position of $c_{2,1}$ (third row, second col) is

$$\begin{bmatrix} 1 & 1 & x & x+1 \end{bmatrix} \begin{bmatrix} c_{0,1} \\ c_{1,1} \\ c_{2,1} \\ c_{3,1} \end{bmatrix} = c_{0,1} + c_{1,1} + x c_{2,1} + (x+1) c_{3,1},$$

where all computations are to be done in $\mathrm{GF}(2^8)$.

AddRoundKey

The **AddRoundKey** layer simply xors the current 128 bit state with a 128 bit round key.

The **key schedule** for AES-128 is as follows. Like for the states, arrange the original 16 byte AES key k in a 4×4 matrix with columns $W(0), W(1), W(2), W(3)$.

The ith round key is then obtained from the matrix with columns $W(4i), W(4i+1), W(4i+2), W(4i+3)$, where $W(4), W(5), \ldots$ are recursively constructed:

$$W(i) = \begin{cases} W(i-4) + W(i-1), & \text{if } 4\nmid i, \\ W(i-4) + \tilde{W}(i-1), & \text{if } 4|i. \end{cases}$$

Here, $\tilde{W}(i-1)$ is obtained from $W(i-1)$ as follows:

$$W(i-1) = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} \quad \Longrightarrow \quad \tilde{W}(i-1) = \begin{bmatrix} S(w_1) + x^{(i-4)/4} \\ S(w_2) \\ S(w_3) \\ S(w_4) \end{bmatrix}.$$

Note that w_1, w_2, w_3, w_4 each are bytes. The function S is the ByteSub substitution. Since that substitution is nonlinear, the round keys are constructed from k in a nonlinear manner (unlike in DES).

As usual, the computation $S(w_1) + x^{(i-4)/4}$ happens in $\mathrm{GF}(2^8)$.

ByteSub

The **ByteSub** layer takes each of the 16 bytes y and replaces it with the byte $S(y)$. As in DES, we could simply describe the (invertible) map by a lookup table. However, like the other steps of AES, it has a very simple mathematical description which we'll discuss next time.

Comment. As for the S-boxes in DES, ByteSub can be implemented in hardware as a lookup table. Since we have $2^8 = 256$ inputs, with 1 byte of output each, this table is 256 bytes large. (See Table 5.1 in our book.)

For comparison, each of the eight S-boxes in DES occupies 2^6 times 4 bits, which is 32 bytes. In total, these are also 256 bytes.

[In contrast to DES it is the case (and necessary for decryption!) that different inputs have different outputs.]

Example 143. (bonus!) $p = 29137$ is an example of a left-truncatable prime: the number itself as well as all truncations 9137, 137, 37, 7 are prime. By simply exhausting all possibilities (start with a single digit and keep adding (nonzero) digits on the left until no choice results in a prime), we find that there is a largest left-truncatable prime, namely, 357686312646216567629137.

https://www.youtube.com/watch?v=azL5ehbw_24

Challenge. Find the largest left-truncatable prime which does not have 1 as a digit.

Send me the prime, and an explanation how you found it, by next week for a bonus point!

Comment. You can play the same game in bases different from 10. We expect that (based on the prime number theorem), for every base, there always are just a finite number of truncatable primes (an extra bonus if you can point me to a proof of that claim!), though the number tends to increase with larger bases. The largest truncatable prime for base 30, for instance, is not known (it is estimated to have about 82 digits in base 30).

https://oeis.org/A103463

Armin Straub
straub@southalabama.edu