Overview of AES

Justin Wyss-Gallifent

April 10, 2022

0.1 Introduction

During the 1990 when DES was being cracked more easily, NIST called for proposals to replace it. There were several very good submissions but the chosen one was by two Belgian Cryptographers, Joan Daemen and Vincent Rijmen. Their cipher was originally called the Rijndael Cipher and three specific variants were chosen for AES.

0.2 Input-Output Basics

AES uses:

1. A plaintext block of size 128 bits.
2. A key which can be 128, 192, or 256 bits.

0.3 Comparison with DES

AES and DES have some similarities and some diffences. The major ones are these:

1. AES and DES are both substitution-permutation ciphers.
2. Both work in Rounds. DES has sixteen Rounds whereas AES has 10 Rounds for a 128-bit key, 12 Rounds for a 192-bit key, and 14 Rounds for a 256-bit key.
3. DES is a Feistel Cipher (each Round manipulates half the input, alternating halves for each Round) whereas AES manipulates the entire input at every Round.
4. Both use S-Boxes, but the S-Box for AES has a mathematical expression, albeit nontrivial.
5. Unlike DES, decryption for AES is not as straightforward as simply reversing the key order, but it is not difficult and AES is still symmetric.

0.4 Funtional Basics

In high-level practical terms, the algorithm works as follows. We'll look at the 128-bit key version, the others are tweaks of this.

0.4.1 Round Key Generation

1. The 128-bit key is broken into 16 bytes which are entered into a 4×4 matrix with one byte per entry ($4 \cdot 4 \cdot 8 = 128$).

   Thus if we have the following key, where each entry is a byte:

   $$B_1 B_2 ... B_{16}$$

   would be placed into:

   $$W = \begin{bmatrix} B_1 & B_5 & B_9 & B_{13} \\ B_2 & B_6 & B_{10} & B_{14} \\ B_3 & B_7 & B_{11} & B_{15} \\ B_4 & B_8 & B_{12} & B_{16} \end{bmatrix}$$

2. This matrix is then recursively grown to the right, adding 40 more columns. This recursive growth is not simple to explain and involves fairly complicated S-Box-related calculations in the field $GF\left(2^8\right)$ (see below).

3. The 10 Round Keys (subkeys) are selected as various 4×4 submatrices. Round Key i consists of columns $4i$, $4i + 1$, $4i + 2$, and $4i + 3$.

0.4.2 Encryption

First the 128-bit plaintex is broken up and placed into a 4×4 matrix as was the key. The entries are treated as elements of the field $GF\left(2^8\right) = GF(256)$ which is isomorphic to the field extension $\mathbb{Z}_2[x]/\left\langle x^8 + x^4 + x^3 + x + 1\right\rangle$. If you're not sure what this is (it's covered towards the end of MATH403), that's fine, just understand that these entries (bytes) can be added via XOR and can be multiplied (but this is a bit more confusing).

Encryption is divided into ten rounds and each round has four layers.

Before embarking on these rounds we first simply add (XOR) the key as-is to the plaintext matrix.

From then on each round has four layers. Working with a 16×16 matrix M we:

1. The ByteSub Transformation (BS):

   Each byte $abcdefg$ in the matrix is broken into $abcd$ and $efgh$ and a lookup is done in a 16×16 S-Box (not listed). The entries in the S-Box are also bytes which means we generate a new 16×16 matrix with byte entries. The S-Box is actually not necessary in that the entries in the S-Box are the results of a rather complex mathematical process involving matrix multiplication and vector addition in $GF\left(2^8\right)$ which itself involves the multiplicative inverse of a byte, which makes sense treating it as an element in $GF\left(2^8\right)$, which is a field. Trappe and Washington have more details on this. the result is then $BS(M)$.

2. The ShiftRow Transformation (SR):

   We take the matrix and rotate row i left by i entries. This is not a logical bit-shift, rather this is a shift of entries of the matrix. The result is then $SR(BS(M))$.

3. The MixColumn Transformation (MC):

   The result is then left-multiplied by a very specific matrix which mixes the columns up a bit. This matrix is:

$$\begin{bmatrix} 00000010 & 00000011 & 00000001 & 00000001 \\ 00000001 & 00000010 & 00000011 & 00000001 \\ 00000001 & 00000001 & 00000010 & 00000011 \\ 00000011 & 00000001 & 00000001 & 00000010 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}_2$$

   This step mixes up each column independently of the other columns. The result is then $MC(SR(BS(M)))$.

4. AddRoundKey (ARK):

   We add (XOR) the Round Key. The result is then $ARK(MC(SR(BS(M))))$.

These four layers make up a Round and there are ten rounds, once with each Round Key.

0.4.3 Decryption

To decrypt we reverse process. This involves finding inverses of BS, SR, MC, and ARK. These are not difficult:

1. The inverse of BS is another lookup table.
2. The inverse of SR just rotates to the right.
3. The inverse of MC uses the inverse of the matrix given.
4. The inverse of ARK is just ARK.

0.5 Notes

A few closing notes.

1. The S-Box was specifically designed to be a mathematical calculation in order to avoid any suspicion that something sneaky was going on with backdoors etc. This issue had haunted DES.
2. The ShiftRow step was added to resist two specific types of attacks which are beyond the scope of our knowledge right now. They are truncated differentials and the Square attack.
3. The MixColumn step diffuses the bytes all over the column.
4. The ten keys are nonlinearly related due to the nature of the recursive construction.
5. Brute force attacks exists which can manage up to six Rounds. Ten was set for the minimum to give a margin of safety.
6. Criteria for development included low RAM requirements and high speed.
7. Some attacks on AES are:
   - Meet-in-the-Middle Attacks: Imagine a cipher that works in two identical stages but with different keys, first encrypting with E_1 and then E_2, then decrypting with D_2 and then D_1. It follows that for a known plaintext P and corresponding ciphertext C that $C = E_2(E_1(P))$ and so $D_2(C) = E_1(P)$. By testing possible keys k_2 and k_1 and comparing $D_2(C)$ and $E_1(P)$ we can, faster than brute force, look for possible k_1 and k_2 which work.
   - Side-Channel Attacks: Rather than cracking the algorithm, look at issues with the technology it is implemented on. Examples are cache attacks, time attacks, and acoustic attacks.